UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES,
ESPECIALIDAD EN TELEMÁTICA

PROYECTO FINAL DE CARRERA

MÓDULO DE TRANSCRIPCIÓN VoIP-TEXTO PARA UNA PLATAFORMA DE INTERCEPTACIÓN LEGAL DE COMUNICACIONES

AUTOR: JESÚS VALLINOT SÁNCHEZ

TUTOR: MANUEL URUEÑA PASCUAL

RESUMEN

En este proyecto se estudia, implementa y evalúa un módulo de transcripción VoIP-texto para una plataforma de interceptación legal de comunicaciones.

Esta plataforma consiste en un software de análisis forense, llamado Xplico, que permite analizar el tráfico de red. Es software libre, y permite clasificar por categorías (tipo de contenido) la información capturada de la red. Ejemplos de estas categorías pueden ser páginas web, imágenes, correo, llamadas VoIP, intercambio de archivos, etc.

El objetivo principal del proyecto es ahorrar tiempo y trabajo a los analistas de la Policía en la tarea de analizar conversaciones de voz sobre IP (*VoIP – Voice over IP*) llevadas a cabo por sospechosos, que estén siendo monitorizados por orden judicial.

De modo que el proyecto consiste, a grandes rasgos, en la incorporación de un software de reconocimiento de voz open source (Sphinx-4) a la plataforma descrita anteriormente. Como se ha señalado, Xplico es capaz de interceptar flujos de datos VoIP, pero hasta ahora, no era posible procesar esos datos automáticamente. Gracias a la incorporación del software de Sphinx-4, el sistema es capaz de transcribir automáticamente las conversaciones interceptadas, proporcionando además, información detallada sobre los interlocutores que intervienen en la conversación. Además de lo citado anteriormente, se ofrece la posibilidad de entrenar el sistema con el objetivo de obtener mejores resultados (mayor precisión en la transcripción) en el futuro.

ABSTRACT

In the present project, VoIP-text transcription module for a legal interception of communications platform has been studied, implemented and evaluated.

This platform is forensic analysis software, called Xplico, to analyze network traffic. It’s free software, and allows categorizing the information captured from the network. Examples of these categories may be web pages, images, mail, VoIP calls, file sharing, etc.

The main goal of the project is to save time and effort to the police’s analysts in the task of analyzing voice conversations over IP (VoIP - Voice over IP) carried out by suspects who are being monitored by court order.

So the project is, broadly speaking, the addition of an open source speech recognition software (Sphinx-4) to the platform described above. As noted, Xplico is able to intercept VoIP data streams, but so far, it was not possible to process these data automatically. With the addition of Sphinx-4 software, the system can automatically transcribe the conversations intercepted, providing also, details on the speakers involved in the conversation. Besides the above, it offers the possibility to train the system in order to obtain better results (increased accuracy in transcription) in the future.

AGRADECIMIENTOS

Expresar agradecimientos por su ayuda en la realización del proyecto a:

1. Manuel Urueña Pascual, del Departamento de Ingeniería Telemática de la Universidad Carlos III de Madrid. Su ayuda ha sido de gran valor.
2. A la comunidad GNU en general, y al grupo CMU Sphinx en especial, por el desarrollo del software libre que ha hecho posible este proyecto.

INDICE

LISTA DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

1.1. MOTIVACIÓN

Lo que se pretende con el presente proyecto es proporcionar a los cuerpos de seguridad, un sistema de interceptación legal de comunicaciones que les permita ser más eficientes a la hora de llevar a cabo su trabajo de investigación y obtención de pruebas sobre actividades delictivas en Internet.

Debido a la gran complejidad que supone la monitorización y, más aún, el análisis de toda la información recabada, así como la obtención de pruebas en dichas actividades, se hace imprescindible para los cuerpos de seguridad, disponer de herramientas cada vez más sofisticadas para combatir el crimen en la red. De hecho, dichas herramientas tienen que seguir siendo actualizadas continuamente para asegurar su correcto funcionamiento y la inclusión de nuevas funcionalidades.

Para dar solución a las anteriores necesidades, están apareciendo nuevas técnicas y herramientas, como las desarrolladas por el proyecto de investigación INDECT, que es un proyecto de investigación en el área de sistemas de seguridad inteligentes, financiado por la Unión Europea, y que busca dotar a los cuerpos de seguridad europeos con mejores medios para mejorar la seguridad de los ciudadanos y garantizar la confidencialidad de la información. Gracias a ello, para realizar este proyecto no será necesario implementar desde cero una plataforma completa de interceptación legal de comunicaciones, sino que se partirá de desarrollos previos. Bastará con añadirle nuevas funcionalidades y/o mejorarla en aspectos concretos.

Teniendo en cuenta lo anterior, se realizó un estudio sobre las distintas posibilidades de ampliación. De entre todas ellas, se decidió llevar a cabo la incorporación de una nueva funcionalidad, basada en el procesamiento automático de flujos de Voz sobre IP (*VoIP – Voice over IP*).

Entre los principales motivos que sustentan esta decisión se encuentran la rápida expansión de las comunicaciones basadas en VoIP y la falta de herramientas que permitan el control de dichas comunicaciones por parte de la Policía.

La siguiente figura representa la estructura básica de un sistema de interceptación legal de comunicaciones. En ella, se muestra a los integrantes que componen el sistema, así como su interrelación:

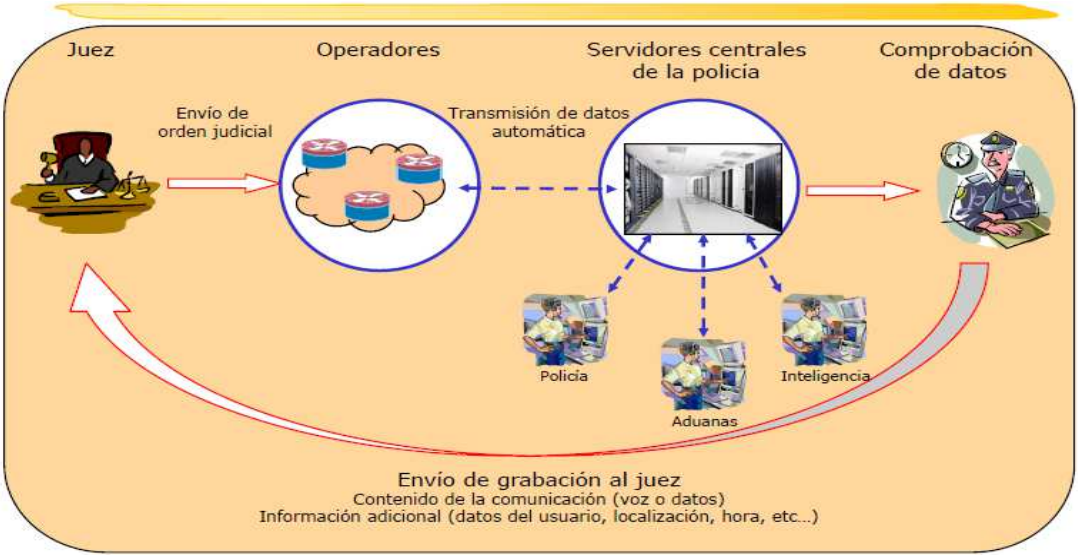

Figura 1.1: Estructura básica de un sistema de interceptación legal de comunicaciones [1]

1.2. OBJETIVOS

El objetivo principal de este proyecto es, por tanto, el de mejorar el sistema de interceptación legal de comunicaciones que está siendo desarrollado para el proyecto INDECT, añadiendo esta nueva funcionalidad para solventar la carencia que presenta el sistema a la hora de monitorizar las comunicaciones de VoIP. Esta funcionalidad consiste en la transcripción pseudo-automática de conversaciones de VoIP, de modo que puedan ser analizadas, pre-clasificadas, e indexadas de manera eficaz.

Aparte de ayudar en la toma de decisiones, también se pretende ahorrar tiempo y recursos a la Policía, ya que esta nueva funcionalidad permitirá automatizar parte de las tareas y reducir el tiempo dedicado a dicha actividad. Hasta ahora, cuando el analista encargado de llevar a cabo la transcripción de la grabación de la conversación se disponía a realizar su labor, no le quedaba más remedio que ponerse a escuchar la conversación de principio a fin e ir transcribiendo cada palabra que oía. Con esta nueva funcionalidad, buena parte del proceso se simplifica, generándose una transcripción inicial automáticamente. Ahora la labor del analista es distinta. Pasa de tener que teclear cada palabra de la conversación a, simplemente, comprobar si la transcripción generada por el sistema es correcta y, en caso de que no lo sea, corregir los errores. Además, para la labor de corrección, dispondrá de herramientas que le facilitarán la labor, permitiéndole ahorrar esfuerzo y tiempo.

Como funcionalidad adicional, se incorpora la disponibilidad de mejorar el sistema haciendo que las transcripciones generadas sean cada vez de mayor calidad, es decir, que disminuya la tasa de error del sistema. Esta funcionalidad de entrenamiento consiste en adaptar el sistema a las voces de los interlocutores habituales que intervienen en las conversaciones. Esto es posible gracias a que, además del audio completo de la conversación, se dispone de los flujos de audio de cada interlocutor que interviene en ella, por separado, dado que se transmiten en flujos RTP diferentes. Estos flujos de audio son entregados al sistema, junto con sus correspondientes transcripciones, de manera que pueda procesarlos y establecer relaciones entre ellos.

1.3. PROYECTO INDECT

El Proyecto INDECT (*Intelligent information system supporting observation, searching and detection for security of citizens in urban environment*) [2] es un proyecto de investigación en el área de sistemas de seguridad inteligentes, financiado por la Unión Europea, en el que trabaja la Policía y universidades de diferentes países de la UE.

A grandes rasgos, el objetivo principal consiste en desarrollar herramientas para mejorar la seguridad de los ciudadanos y garantizar la confidencialidad de la información. Su actuación se centra en detectar amenazas tanto en entornos reales (cámaras inteligentes) como en entornos virtuales (Internet).

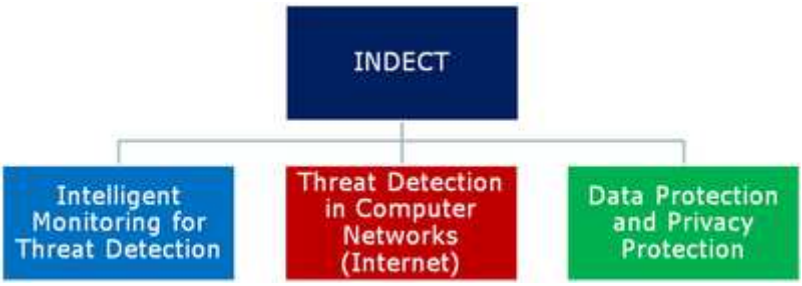

Figura 1.2: Objetivos principales del proyecto INDECT [2]

El consorcio del proyecto está formado por 17 socios, 11 de ellos universidades, entre las que destaca al frente del proyecto la AGH University of Science and Technology (Polonia), bajo la supervisión del profesor Andrzej Dziech, el coordinador del Proyecto. Entre ellas también se encuentra la Universidad Carlos III de Madrid. También participan cuerpos de seguridad como la policía de Irlanda del Norte y la Policía de Polonia. El proyecto comenzó oficialmente en enero de 2009 y su fecha de finalización es diciembre de 2013.

Entrando un poco más en detalle, los objetivos de este proyecto serían tres:

1. Desarrollar un sistema inteligente para la detección automática de amenazas y reconocimiento de comportamientos delictivos:

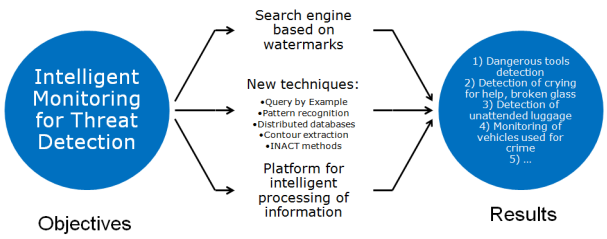

Figura 1.3: Detección inteligente de amenazas [2]

2. Proporcionar herramientas informáticas más eficientes a los cuerpos de seguridad en su lucha contra el crimen en la red. Entre ellas se encuadra la actual plataforma de interceptación legal de comunicaciones.

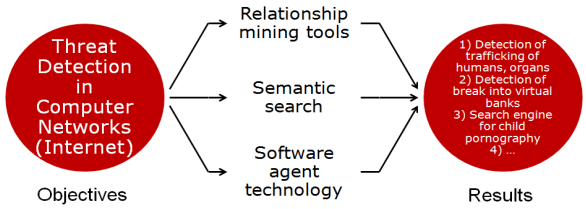

Figura 1.4: Herramientas proporcionadas [2]

3. Desarrollar técnicas que permitan garantizar la privacidad de la información transmitida y/o almacenada, mediante criptografía y firma digital:

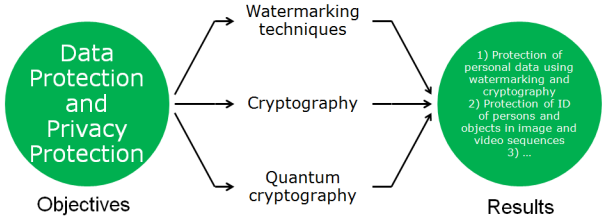

Figura 1.5: Técnicas desarrolladas [2]

El proyecto pone especial énfasis en los aspectos éticos del mismo, garantizando, entre otras cosas, que nunca, durante el proyecto, se llevará a cabo el análisis de información personal sin el consentimiento previo de los individuos afectados.

Entre los organismos encargados de garantizar el cumplimiento de estas normas éticas, se encuentra la policía Norirlandesa, reconocida como una de las más integras y eficientes en su labor de protección de los derechos humanos.

1.4. ESTRUCTURA DE LA MEMORIA

La estructura de la memoria es la siguiente:

1. Introducción: En este capítulo se describe cuales han sido las motivaciones y los objetivos que se persiguen con la realización del proyecto. También se describe el proyecto de investigación INDECT, donde se encuadra este proyecto.

2. Estado del arte: En este capítulo se realizará un estudio sobre las diversas tecnologías disponibles para llevar a cabo los objetivos del proyecto. Mencionar que estas tecnologías deben ser Open Source (código libre), lo cual limita bastante el número de herramientas disponibles así como su calidad.

3. Diseño de la aplicación: Esta parte es la más breve de la memoria, pero a su vez la más descriptiva de todas, ya que es donde se describe la arquitectura funcional de la aplicación. Se analizarán cada uno de los módulos que integran el sistema así como su interrelación con los demás módulos.

4. Implementación de la solución: Esta es la parte más técnica de la memoria. En ella se describe en detalle la funcionalidad de cada módulo desarrollado. En vez de mostrar el

código fuente, se ha preferido realizar una descripción verbal de cada componente y su función. De ese modo, quien quiera comprender de manera general el desarrollo de la aplicación no tendrá más que leer estas líneas, y aquel que quiera comprenderlo a un nivel más profundo (p.e. futuro desarrollador) puede acceder directamente al código de la aplicación.

5. Evaluación de la solución: Una vez desarrollada, la aplicación será sometida a una serie de pruebas que medirán el rendimiento del sistema en términos de eficiencia. Se analizarán aspectos tales como el consumo de recursos, capacidad de adaptación a distintos entornos y locutores, precisión en la transcripción, usabilidad del interfaz gráfico, etc.

6. Conclusiones y trabajos futuros: Se analizarán los resultados obtenidos en comparación con los objetivos iniciales del proyecto, para ver si realmente se ha obtenido una solución satisfactoria que se adapte a los requisitos del problema. Por último, se hará hincapié en aquellos aspectos que deban ser mejorados para conseguir unos resultados óptimos.

CAPÍTULO 2: ESTADO DEL ARTE

2.1. *Voz sobre IP (VoIP)*

VoIP [3] es el acrónimo de "*Voice Over Internet Protocol*" y, tal y como su nombre indica, es una tecnología que permite la transmisión de voz encapsulada en paquetes IP sobre redes de datos como puede ser Internet. Por lo tanto, supone un gran avance, al permitir transmitir voz y datos a través del mismo medio compartido, con el consiguiente ahorro económico que ello supone. A su vez, es importante distinguirla de la telefonía IP, ya que esta última es mucho más amplia y se basa en la integración y expansión de los servicios de telefonía tradicionales a través de redes IP.

2.1.1. Protocolos

La siguiente imagen muestra la arquitectura de protocolos de la tecnología VoIP:

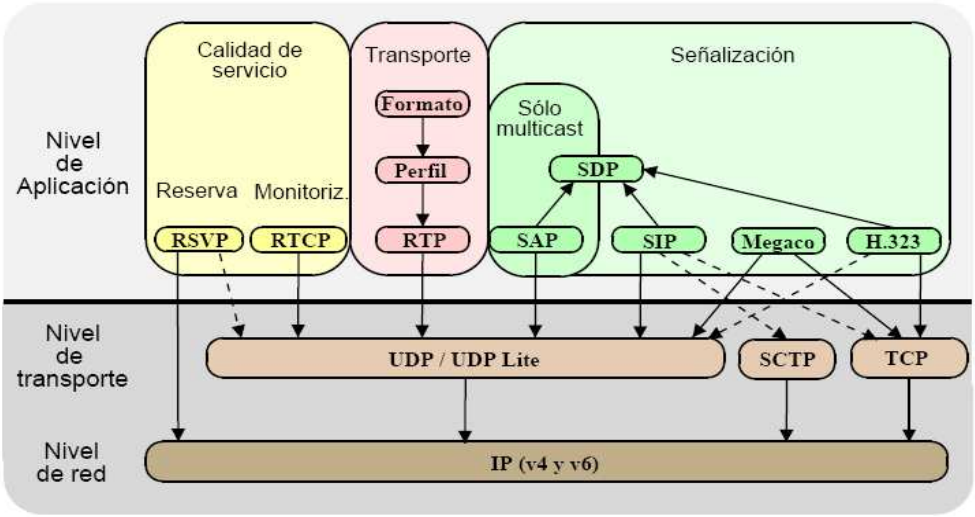

Figura 2.1: Arquitectura de protocolos relacionados con VoIP [3]

Como puede apreciarse, existen multitud de protocolos involucrados con la tecnología VoIP. De entre todos ellos, se describirán los que interactúan de manera especial con el módulo de transcripción, a la hora de proveer los flujos de audio interceptados a través de la red, a saber: SIP, RTP, RTCP y SDP.

- ***Session Initiation Protocol (SIP)***

  SIP [4] es, actualmente, el principal protocolo utilizado para llevar a cabo la señalización en redes VoIP. Al ser un protocolo de señalización, sólo gestiona el establecimiento, control y finalización de las sesiones de comunicación. Destaca por su simplicidad, escalabilidad y facilidad para integrarse con otros protocolos y aplicaciones.

  Mencionar que es un protocolo de nivel de aplicación y que puede transportarse tanto sobre UDP como sobre TCP. Los clientes SIP usan el puerto 5060.

En caso de utilizar un protocolo seguro como SIPS, se utiliza el puerto 5061. Utiliza mensajes de texto similares a los de los protocolos HTTP y SMTP.

Destacar también que, en el año 2000, SIP se convirtió en el protocolo de señalización de 3GPP, permitiendo la implementación de la arquitectura IMS (*IP Multimedia Subsystem*), que favorece la convergencia fija-móvil.

A continuación se describen cada uno de los componentes que integran la arquitectura SIP:

- Agentes de usuario (UA): Son utilizados por los usuarios para establecer sesiones. Se integran en cada extremo de la comunicación, y se encargan de emitir y recibir los mensajes SIP. Pueden actuar como clientes (UAC: *User Agent Clients*), cuando realizan una petición, o como servidores (UAS: *User Agent Servers*), cuando reciben peticiones. Por eso, cada UA debe implementar ambas modalidades.

- Servidores de registro: Permiten ubicar en qué punto de la red está conectado un usuario (i.e. la dirección IP del UA). Para ello, se utiliza el mecanismo de Registro, cuyo funcionamiento será descrito en la sección siguiente.

- Servidores Proxy y de redirección: Un mismo servidor puede redirigir o actuar como Proxy dependiendo de la situación. Su función es la de encaminar los mensajes generados por un UAC hacia un UAS. Para ello, pueden actuar de dos maneras:

    1. Como Proxy, encaminando el mensaje de un extremo a otro, y formando parte del camino entre ambos.
    2. Como *Redirect*, generando una respuesta que indica al UAC la dirección del destino o de otro servidor que le guíe hacia su destino. Pero una vez hecho esto, no vuelve a intervenir en la comunicación.

En lo que se refiere al funcionamiento del protocolo, en la siguiente figura se muestra un ejemplo de establecimiento de llamada mediante el protocolo SIP:

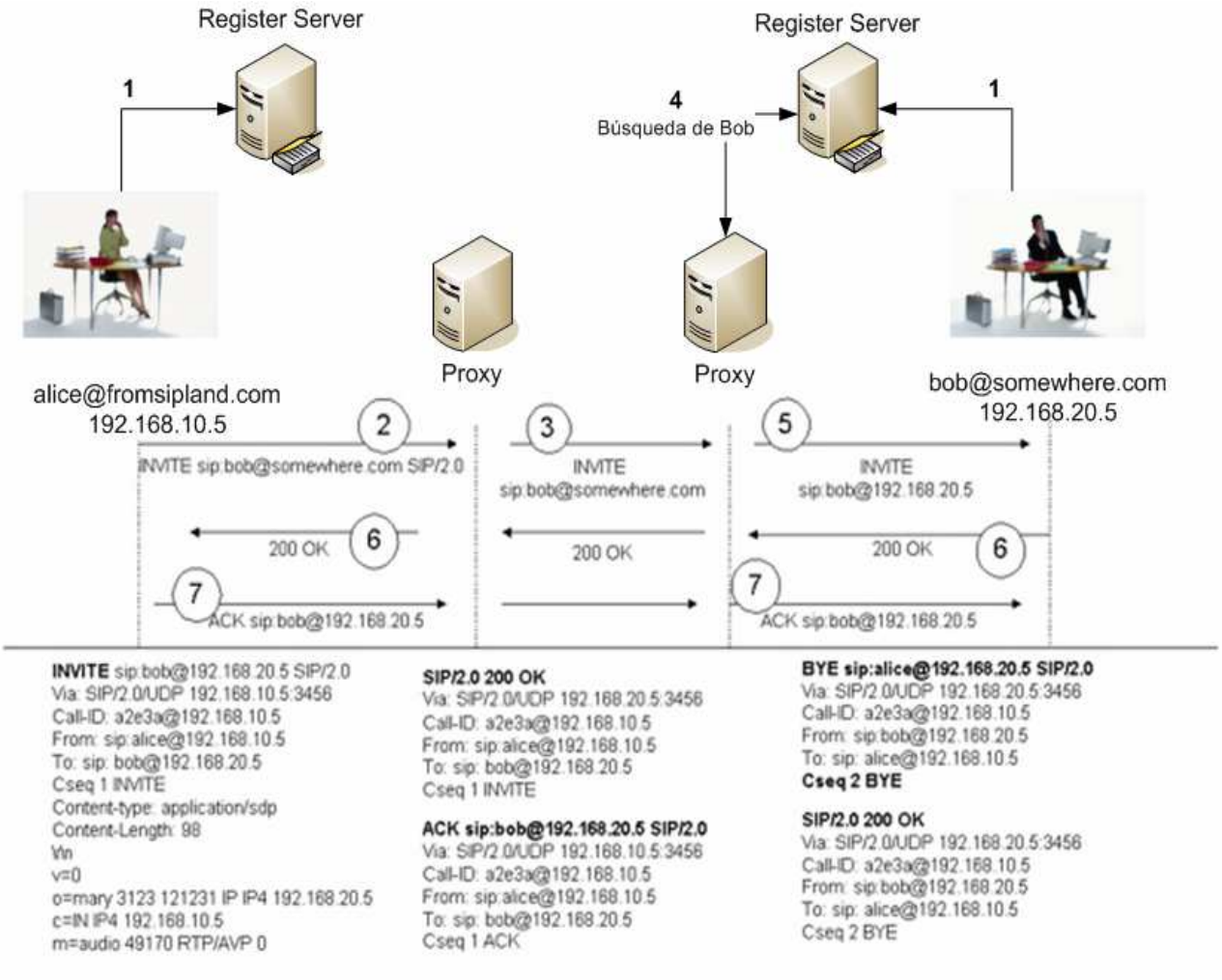

Figura 2.2: Establecimiento de llamada mediante el protocolo SIP [3]

Cada usuario tiene asociada una dirección lógica, que es independiente de su ubicación física dentro de la red. Esta dirección SIP tiene el siguiente formato: *sip:usuario@dominio.* Sin embargo, la dirección física (dirección IP) si depende de la ubicación del usuario.

Paso 1: Lo primero que han de realizar los usuarios que intervienen en la comunicación, en este caso Alice y Bob, es iniciar sus respectivos terminales. En ese momento, el UA enviará una petición REGISTER para darse de alta en su correspondiente servidor de registro, informando de a qué dirección física (IP) debe asociarse su dirección lógica (SIP). El servidor de registro realizará dicha asociación (*binding*). Dicha asociación es temporal, y debe renovarse en caso de querer conservarla.

Paso 2: En el momento en que Alice desee contactar con Bob, realizará una petición *INVITE* hacia su Proxy, que será el encargado de encaminar el mensaje. Para poder comunicarse con el Proxy de Bob, antes deberá hacer una consulta al servidor DNS que sirva el dominio de la dirección de Bob (en este caso, *somewhere.com*), para obtener su IP.

Paso 3: El Proxy de Alice lanza un INVITE hacia el Proxy de Bob, que reenvía la petición hacia el destinatario.

Pasos 4: Para ello, el Proxy de Bob tendrá que consultar con anterioridad la dirección de Bob en su servidor de registro.

Paso 5: Una vez que el Proxy de Bob conoce la dirección IP de éste, lanza un INVITE contra su equipo.

Paso 6: Cuando Bob descuelga el teléfono, se retransmite un mensaje *ACK 200 OK* hacia emisor de la llamada.

Paso 7: Se envía un ACK con la dirección SIP del destino, desde el origen. En este punto, la conexión queda establecida y se procede al envío de datos. Estos datos se intercambian directamente entre los extremos mediante RTP. Para terminar, cuando un usuario cuelgue se enviará un mensaje *BYE* que dará por finalizada la conexión.

- ***Real-Time Transport Protocol (RTP)***

En aplicaciones de Voz sobre IP, RTP [5] es el responsable de la transmisión de los datos, mientras que SIP se encarga únicamente de la señalización y el establecimiento de llamada. Su función principal consiste en multiplexar distintos flujos de datos en un solo flujo de paquetes UDP, todo ello en tiempo real, pudiéndose enviar a uno (*unicast*) o varios destinos (*multicast*).

Es un protocolo de nivel de sesión y se ejecuta sobre UDP, ya que el objetivo principal es conseguir la máxima velocidad posible a la hora de transmitir los datos, aunque se pierdan paquetes. Por lo tanto, no garantiza la entrega del paquete en recepción, es decir, no es fiable. No ofrece control de flujo, ni de errores, y no está orientado a conexión. Utiliza números de secuencia para reorganizar los paquetes, en caso de que lleguen en desorden al destino, y marcas de tiempo (*timestamps*) para ajustar los intervalos de muestreo de acuerdo a la secuencia original.

- ***Real Time Control Protocol (RTCP)***

RTCP [6] ofrece un mecanismo de control al protocolo RTP. Su función principal consiste en proporcionar información sobre la calidad de servicio (*QoS*) ofrecida por RTP. Para ello, genera estadísticas sobre la conexión y transmisión de los datos para que las aplicaciones puedan mejorar la calidad de servicio.

- ***Session Description Protocol (SDP)***

La función principal de SDP [7] consiste en proporcionar información sobre sesiones de comunicación multimedia, como por ejemplo, la negociación de parámetros. No interviene en la entrega de los datos sino que se encarga de gestionar la negociación entre las entidades que intervienen en la sesión. Los parámetros que gestiona se denominan *perfil de sesión*.

2.1.2. Códecs

La voz ha de codificarse para poder ser transmitida por la red IP. Para ello, se hace uso de *Códecs* (***co***dificador-***dec***odificador) que permitan la codificación y compresión de los

datos en el origen, y su posterior decodificación y descompresión en el destino. Según el Códec utilizado en la transmisión, se consumirá un determinado ancho de banda. La cantidad de ancho de banda utilizada suele ser directamente proporcional a la calidad de los datos transmitidos.

De entre todos los códecs utilizados en VoIP se describirá el G.711 [8], ya que es el que reúne las características exigidas por el módulo de transcripción, a la hora de decodificar los flujos de audio interceptados a través de la red. Estas características son:

1. PCM (*Pulse Code Modulation*): La modulación por impulsos codificados es un procedimiento de modulación utilizado para transformar una señal analógica en una señal digital. Consiste en tres pasos: Muestreo, cuantificación y codificación.
2. Tasa de muestreo de 8 kHz: El proceso de muestreo consiste en tomar valores instantáneos de una señal analógica, en intervalos de tiempo iguales. Los valores instantáneos obtenidos se denominan muestras. Esta tasa de muestreo es la habitual en telefonía y voz sobre IP.

Además de las características anteriores, destacar que posee una tasa de bit de 64 Kbps, y dos versiones, *u-law* (usada en EEUU y Japón) y *a-law* (usada en Europa).

2.1.3. Ventajas

A continuación se enumeran las principales ventajas de la tecnología VoIP frente a la telefonía convencional:

1. La ventaja principal es el coste de la comunicación, ya que una llamada realizada a través de VoIP es, en la mayoría de los casos, mucho más barata que una realizada a través del teléfono, especialmente en el caso de llamadas internacionales.
Esto se debe, principalmente, a que se utiliza la misma red para la transmisión de datos y voz. Generalmente las llamadas entre dos teléfonos IP son gratuitas, mientras que si se realiza una llamada de un teléfono IP a un teléfono convencional, el coste de la comunicación es inferior y se le cobra al usuario del teléfono IP.

2. En la mayoría de los casos, durante la comunicación hay una persona que habla y otra que escucha, lo que supone que una parte de la conexión telefónica está siendo desaprovechada constantemente. Según esto, el ancho de banda dedicado a la comunicación podría reducirse a la mitad, manteniendo la calidad de ésta. Además de esto, también se dan situaciones en las que ninguna de las dos personas habla. En esos casos también podría reducirse el ancho de banda, sin perder calidad, para aprovechar mejor los recursos. Por lo tanto, la multiplexación estadística de la conmutación de paquetes es una de las ventajas que ofrece la tecnología VoIP, frente a la reserva de recursos de conmutación de paquetes ofrecida por la telefonía tradicional.

3. La mayoría de los proveedores de VoIP ofrecen servicios adicionales gratuitos, incluidos en el servicio estándar, por los cuales, los operadores de telefonía convencional cobran tarifas adicionales. Entre esos servicios se incluyen la identificación de llamadas, servicio de llamadas en espera, enviar la llamada directamente al correo de voz, etc.

2.1.4. Inconvenientes

Los principales inconvenientes que dificultan la expansión de la tecnología VoIP se deben a limitaciones tecnológicas. Por lo tanto, es bastante probable que, una vez que se solucionen dichas limitaciones, la tecnología VoIP acabe imponiéndose. A continuación se describen algunos de los principales inconvenientes:

1. Para llevar a cabo la comunicación VoIP es necesario disponer de una conexión de banda ancha. Aunque el despliegue de la banda ancha se ha acelerado en los últimos años, aún quedan zonas geográficas donde no se dispone de ella. Además, hay que tener en cuenta que para establecer una comunicación VoIP de calidad entre dos nodos, es necesario que ambos dispongan de banda ancha, no solo uno de ellos.

2. Es necesario disponer de conexión eléctrica. Esto que se da por sentado, puede provocar problemas en caso emergencias durante cortes eléctricos. En estos casos, los teléfonos convencionales siguen funcionando, ya que la alimentación llega a través del propio cable telefónico, mientras que los teléfonos VoIP quedan inhabilitados, a no ser que estén conectados a una UPS (alimentación ininterrumpida).

3. Al utilizarse, en VoIP, una conexión de red para llevar a cabo la comunicación, es posible que ésta se vea afectada por problemas transitorios como retardos elevados en la transmisión o pérdida de paquetes. Debido a ello, pueden surgir interferencias o, incluso, puede cortarse la comunicación.

4. Al realizarse la comunicación a través de Internet, pueden surgir problemas de seguridad. Por ejemplo, los paquetes de datos que contienen la voz codificada pueden ser interceptados por una tercera parte (como es el caso del presente proyecto). No obstante, hay herramientas como el cifrado, que permiten defenderse de la mayoría de los ataques.

2.2. XPLICO

Xplico [9] es una herramienta de análisis forense, diseñado con el objetivo de capturar y decodificar el tráfico de red, mostrando la información al usuario a través de un interfaz web.

Existen otras herramientas similares, como Wireshark o tcpdump, que también permiten capturar y decodificar el tráfico de red, pero la información que proporcionan al usuario es de bajo nivel, es decir, información muy detallada sobre los protocolos de red. Xplico trabaja a alto nivel proporcionando información sobre el contenido intercambiado en lugar del protocolo empleado.

El funcionamiento de Xplico es sencillo: una vez obtenido un fichero de tráfico de red, es posible extraer, clasificado por categorías de aplicación, todos los contenidos intercambiados. Así, por ejemplo, de un fichero de captura *.pcap*, Xplico decodificará los correos, páginas web, llamadas VoIP, sesiones SFTP/FTP, etc.

Creado por Gianluca Costa & Andrea de Franceschi a comienzos del año 2007, sigue en desarrollo en la actualidad.

Este software se distribuye bajo licencia GPL de código abierto y sólo es compatible con el sistema operativo Linux.

2.2.1. Arquitectura

El sistema Xplico se compone de cuatro macro-componentes:

1. Un gestor llamado Dema (*Decoder Manager*), encargado de:

    - Organizar los datos de entrada
    - Configurar el decodificador y los *handlers* (aplicaciones encargadas del tratamiento de la información)
    - Lanzar el decodificador y los *handlers*
    - Controlar la ejecución del decodificador y los *handlers*

2. Un decodificador de red llamado Xplico

    - Se compone de tres tipos de módulos:

        1. Módulos de captura
        2. Módulos de procesamiento (dissectors)
        3. Módulos de distribución (dispatcher)

    - Los datos de entrada (*raw data*) llegan a través de los módulos de captura. Después, los datos viajan hacia los módulos de procesamiento. Por último, los datos ya estructurados son enviados a los módulos de distribución para hacerlos llegar hacia las aplicaciones que lo requieran.

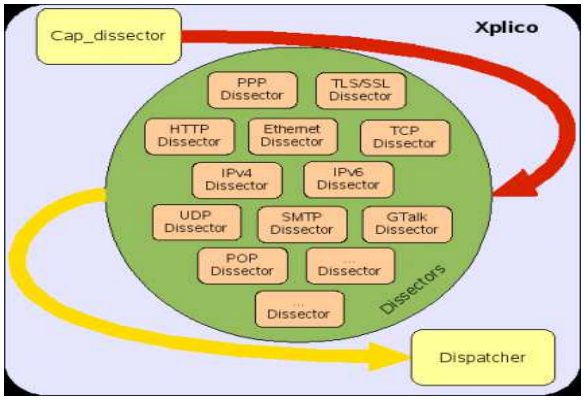

Figura 2.3: Interrelación entre los distintos módulos del decodificador de red [9]

3. Gestores de los datos decodificados (*Handlers)*

4. Un interfaz gráfico desarrollado en CakePHP

La siguiente figura ilustra la arquitectura de Xplico:

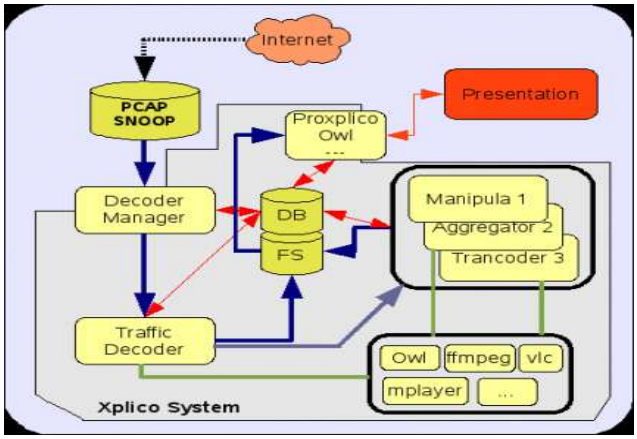

Figura 2.4: Arquitectura de Xplico [9]

Xplico se ha centrado en los protocolos más usados en la actualidad. Esto se consigue mediante la implementación de módulos llamados *dissectors* que se encargan de analizar y reconstruir los datos recibidos (*raw data*). Algunos ya están totalmente implementados y otros todavía están en desarrollo. En la siguiente figura se muestran los *dissectors* ofrecidos por Xplico así como su nivel de desarrollo:

Dissector	Status	Note	Dissector	Status	Note
ARP	90%	—	PJL	90%	—
Radiotap	90%	—	NNTP	95%	—
Ethernet	100%	—	MSN	60%	v1 beta
PPP	90%	—	IRC	85%	—
VLAN	95%	—	YAHOO	0%	—
L2TP	70%	—	GTALK	0%	—
IPv4	98%	—	EMULE	0%	—
IPv6	98%	—	SSL/TLS	0%	with keys
TCP	95%	—	IPsec	0%	with keys
UDP	100%	—	802.11	60%	no encryp.
DNS	80%	—	LLC	60%	—
HTTP	100%	—	MMSE	95%	over HTTP
SMTP	95%	—	Linux cooked	95%	SLL
POP	95%	—	TFTP	90%	—
IMAP	95%	—	SNOOP	100%	Format
SIP	80%	—	PPPoE	90%	—
RTP	70%	—	Telnet	90%	—
RTCP	60%	—	WebMail	90%	—
SDP	70%	—	Paltalk Exp.	60%	—
FB chat	90%	—	Paltalk	90%	—
FTP	90%	—	NetBIOS	5%	Ses. Mes.
IPP	90%	—	SMB	0%	—

Figura 2.5: Protocolos soportados por Xplico y su nivel de desarrollo [9]

2.2.2. Ejemplo de aplicación

A continuación se muestra, a modo de ejemplo, el proceso de análisis de paquetes VoIP mediante Xplico:

1. Conectarse al interfaz web de Xplico.

2. En esta primera imagen se muestra la ventana principal de Xplico. A la izquierda se encuentra un menú con *Cases* (casos). Se puede elegir entre crear un nuevo caso o seleccionar uno de los que aparecen en la lista (en el centro de la pantalla). Estos proyectos pueden generarse a través de ficheros *.pcap* (ficheros que contienen información sobre los paquetes intercambiados, protocolos usados, conexiones establecidas, etc), o bien a través de capturas en vivo (*live*) realizadas a través de un interfaz de red.

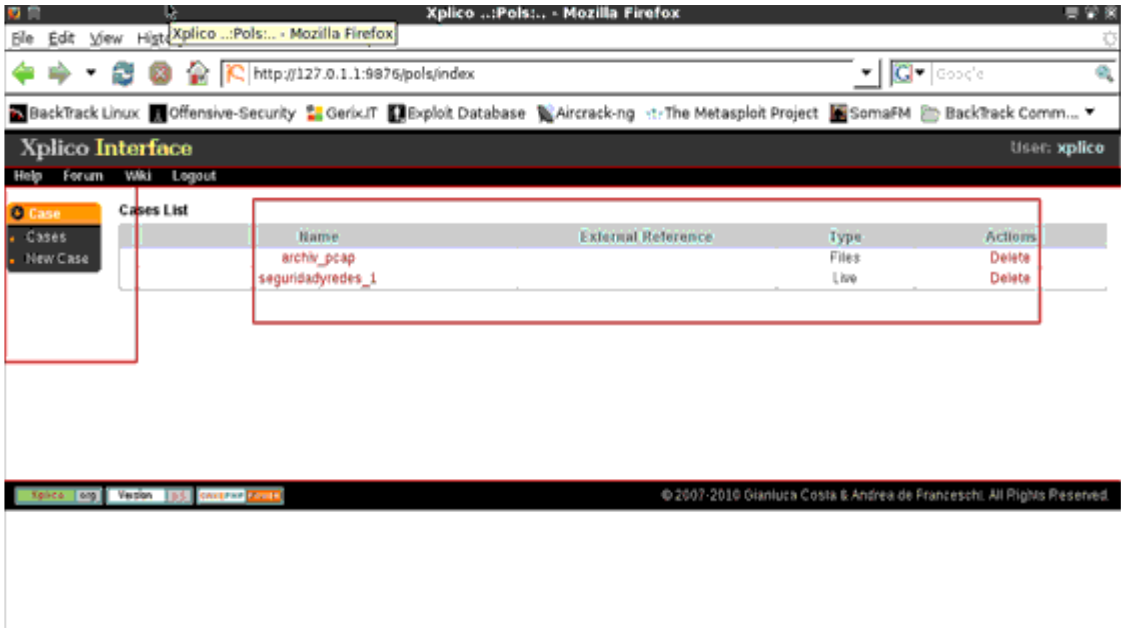

Figura 2.6: Ejemplo de "Cases" disponibles en Xplico [9]

3. Si se selecciona un proyecto generado a través de *.pcap*, se obtendría algo similar a la siguiente vista. A la izquierda se pueden observar los distintos protocolos y herramientas disponibles. En el resto de la imagen se muestran los paquetes capturados para los distintos protocolos.

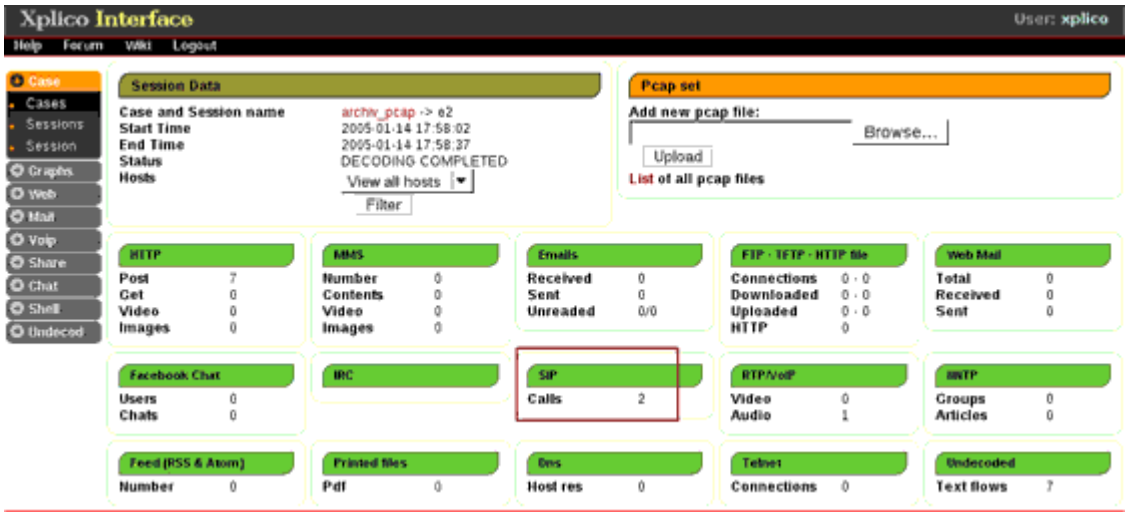

Figura 2.7: Pantalla de ejecución de Xplico [9]

5. Si se selecciona el campo VoIP de la columna de la izquierda, nos llevaría hacia la siguiente vista, en la que se pueden observar las sesiones SIP capturadas.

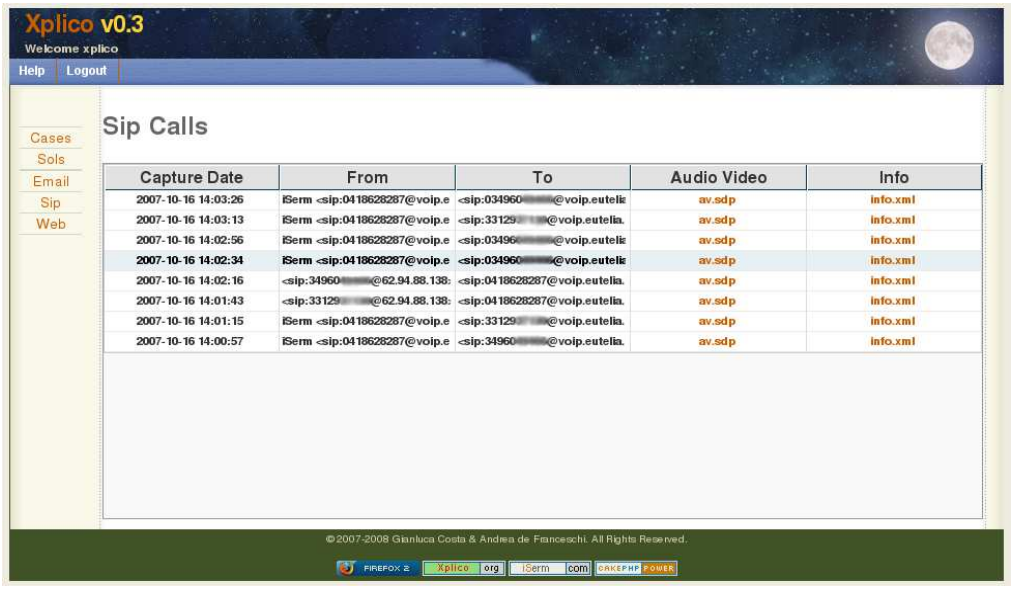

Figura 2.8: Listado de llamadas SIP capturadas en Xplico [9]

5. Si se selecciona cualquiera de las sesiones SIP capturadas, se mostraría la siguiente vista, en la que se puede escuchar el audio de la sesión SIP seleccionada.

Figura 2.9: Audio de una sesión SIP en Xplico [9]

2.2.3. CakePHP

CakePHP es el marco de trabajo utilizado para la implementación de Xplico. Sirve como entorno de desarrollo para un lenguaje de programación rápido como PHP, y es de

código abierto. Gracias a ello se pueden crear aplicaciones web de manera rápida, estructurada y flexible. Además, tiene un equipo de desarrolladores y una comunidad activos, que permiten que el código se actualice y mejore constantemente. Se basa en la arquitectura Modelo-Vista-Controlador.

2.3. SISTEMAS DE RECONOCIMIENTO DE VOZ

Desde que se iniciase la investigación en tecnologías de reconocimiento de voz, allá por la década de los 50, mucho se ha avanzado en este campo. Su evolución va de la mano del desarrollo de la tecnología digital. No obstante, y a pesar de los avances logrados, continúa siendo una tecnología con una tasa de error considerable, debido en gran parte a su complejidad y a la gran cantidad de obstáculos a los que se enfrenta. Sin embargo, no todos los sistemas desarrollados dentro de este campo de investigación son iguales. Existen aplicaciones comerciales que presentan un mayor grado de desarrollo y eficiencia, frente a aquellas que se encuadran dentro del marco Open Source.

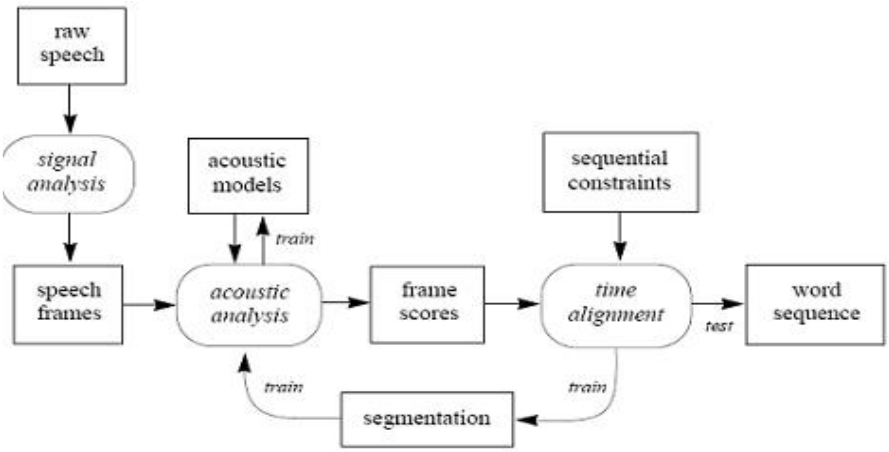

Figura 2.10: Estructura habitual de un sistema de reconocimiento de voz [10]

Los dos factores fundamentales que influyen en el rendimiento de un sistema de reconocimiento de voz son la velocidad de reconocimiento y la precisión. A estos aspectos es a lo que los investigadores han dedicado más tiempo desde que comenzaron a desarrollarse los primeros sistemas, y seguirán siendo los que más esfuerzos requieran, pues ellos definen prácticamente la totalidad del rendimiento del sistema.

Otros aspectos que se han mejorado son la disponibilidad de reconocimiento en tiempo real, la flexibilidad del sistema, es decir, la capacidad de reconocer el habla humana independientemente del locutor, y la extensión de modelos lingüísticos y acústicos más extensos.

El impacto de todas estas mejoras ya se ha hecho notar, permitiendo a los usuarios disfrutar de mejores prestaciones y sistemas más completos y cómodos de manejar.

2.3.1. Clasificación

Los sistemas de reconocimiento de voz pueden clasificarse según los siguientes criterios:

- Capacidad de adaptación: determina si el sistema permite ser entrenado y, de ese modo, mejorar sus prestaciones, o por el contrario, no permite modificaciones.
- Dependencia del interlocutor: determina si el sistema está diseñado para reconocer la voz de un solo interlocutor, o de varios.
- Continuidad: Hay sistemas en los que el interlocutor puede hablar de manera fluida, y otros en los que el interlocutor debe hacer pausas entre palabra y palabra.
- Robustez: determina si el sistema está diseñado para usarse con señales poco ruidosas o, por el contrario, puede funcionar aceptablemente en entornos ruidosos.
- Tamaño del léxico: determina si el sistema está diseñado para reconocer palabras de un léxico reducido (unos cientos de palabras) o extenso (miles de palabras).

Como se verá más adelante, el tamaño del léxico es, probablemente, la característica más notoria a la hora de decantarse entre unos sistemas u otros. De este modo, los distintos sistemas pueden agruparse en dos categorías principales:

- Sistemas de dictado automático: Estos sistemas son los más complejos y permiten reconocer grandes cantidades de palabras y expresiones, además de varios interlocutores distintos. Para nuestra aplicación es lo más indicado.

- Sistemas de control de aplicaciones mediante comandos: Son los más habituales ya que son más simples. Tienen capacidad de reconocimiento limitado tanto en vocabulario como en el número de interlocutores.

2.3.2. Arquitectura

Un aspecto importante a la hora de diseñar un sistema de reconocimiento de voz es la elección del tipo de aprendizaje que se va a utilizar para desarrollar el sistema. Básicamente, existen dos tipos:

- Aprendizaje deductivo: El sistema mejora a través de los conocimientos que le transfiere un humano.

- Aprendizaje inductivo: El sistema puede, automáticamente, conseguir los conocimientos necesarios a partir de muestras existentes sobre la tarea que se desea modelizar. Ejemplo de esto son los sistemas basados en Modelos Ocultos de Markov (HMM – Hidden Markov Model) o las redes neuronales artificiales.

En la práctica, suelen combinarse ambos aprendizajes para, de ese modo, adaptarse mejor a un mayor número de situaciones. De hecho, la mayoría de los sistemas de reconocimiento de voz modernos están basados en el Modelo Oculto de Markov.

2.3.3. Aplicaciones

Las aplicaciones de esta tecnología son ilimitadas y, a medida que se mejora la tecnología, aumenta aún más el número de aplicaciones. Las aplicaciones más destacadas son:

- Sistemas portátiles, como por ejemplo los teléfonos móviles o las tablets, los cuales presentan restricciones de tamaño y forma, de manera que la voz se convierte en una solución práctica para introducir datos en estos dispositivos.

- Sistemas diseñados para discapacitados: Los sistemas de reconocimiento de voz pueden ser útiles para personas con discapacidades que les impidan teclear con fluidez, así como para personas con problemas auditivos, que pueden usarlos para obtener texto escrito a partir de señales de audio. Esto permitiría, por ejemplo, que los aquejados de sordera pudieran realizar llamadas telefónicas.

- Aplicaciones en el campo de la salud y la telemedicina, como por ejemplo, dictado automático de recetas médicas.

- Aplicaciones militares tales como comunicaciones aéreas y marítimas, telecontrol de aeronaves, activación de mandos de control, sistemas de navegación, etc.

- Domótica, con la creación de viviendas inteligentes que mejoren la calidad de vida de las personas.

- En el área de los videojuegos, se puede usar, por ejemplo, para controlar a los personajes del juego.

- Por último, otra de sus aplicaciones puede ser la descrita en este proyecto, es decir, la monitorización, análisis y clasificación de conversaciones llevadas a cabo a través de redes VoIP.

2.3.4. Obstáculos

El reconocimiento automático de voz puede entenderse como el mapeo de una señal continua, la voz humana, hacia una secuencia de muestras discretas (por ejemplo, fonemas, palabras o frases).

Se desea obtener la mejor calidad posible en el proceso, pero el principal obstáculo que se presenta es la enorme variabilidad de la voz humana. Entre en juego un gran número de variables, entre las que destacan:

- La gran variedad de lenguas y dialectos, además de sus variaciones. Cada una tiene su propia semántica, sintaxis, fonología…
- La gran variedad de hablantes de esa lengua concreta, cada uno con unas características fonéticas distintas
- La calidad del canal de comunicación. Un entorno libre de ruido sería lo deseable, pero pocas veces se da esta situación. Además, hay que tener en cuenta distintos tipos de canales. No es lo mismo un canal telefónico o de VoIP sobre Internet, que el canal que comparten dos personas mientras hablan cara a cara.

2.3.5. Modelos acústicos y de lenguaje

- **Modelo acústico**

  Un modelo acústico se crea a partir de:

1. Ficheros de audio que contienen grabaciones de voces de individuos.
2. Ficheros de texto que contienen las correspondientes transcripciones textuales.

El siguiente paso consiste en utilizar un software que, compilando los ficheros anteriores (los cuáles conforman un *speech corpus*), cree representaciones estadísticas de los sonidos que componen cada palabra. Este proceso se conoce como entrenamiento o creación del modelo acústico.

Este modelo estadístico creado es usado, posteriormente, por el software de reconocimiento de voz para realizar el reconocimiento de nuevas grabaciones.

- **Modelo de lenguaje**

Un modelo de lenguaje consiste en un fichero que contiene secuencias de palabras relacionadas con sus distintas probabilidades de aparición. Se utiliza para intentar replicar las propiedades de un lenguaje y predecir la siguiente palabra en una conversación. Se basa en el uso de modelos *N-gramas* (subsecuencia de N elementos de una secuencia dada) que son modelos probabilísticos usados para predecir el siguiente elemento en una secuencia. En concreto, destaca los el uso de *bigramas* (n=2) y *trigramas* (n=3).

  - Fichero de gramática: Fichero mucho más pequeño que contiene una serie de combinaciones de palabras predefinidas. Se utiliza en aplicaciones dedicadas al reconocimiento de un número limitado de palabras. El proceso de reconocimiento es mucho más rápido y preciso, a costa de reducir el vocabulario.

  - *Speech corpus*: Base de datos que contiene ficheros de audio (grabaciones de voz) y de texto (las transcripciones de las grabaciones), traducidos a un formato que pueda ser usado para crear modelos acústicos (los cuáles pueden ser usados, entonces, con un software de reconocimiento de voz).

2.3.6. Tasa de muestreo, códecs y canal de comunicación

- ***Tasa de muestreo***

El audio puede ser codificado a distintas tasas de muestreo (las más comunes: 8, 16, 32, 44.1, 48 y 96 kHz) y diferente número de bits por trama (lo más común: 8, 16 y 32-bits). Los motores de reconocimiento de voz funcionan mejor si el modelo acústico que usan fue entrenado con audio con la misma tasa de muestreo y bits por trama que la voz utilizada en el proceso de reconocimiento.

- ***Reconocimiento de voz en telefonía y en VoIP***

La limitación principal es el ancho de banda al cual puede ser transmitida la voz. Una línea telefónica digital convencional tiene una tasa binaria de 64 kbits/seg, con una tasa de muestreo de 8 kHz y 8-bits/trama (8000 tramas/seg * 8 bits/trama = 64000 bits/seg). Por lo tanto, se necesitan modelos acústicos entrenados con ficheros de audio (voz) de 8 kHz/8-bits. Los códecs (encargados de codificar/decodificar la señal o el flujo de datos digitales) determinan la tasa de muestreo/bits por trama de la transmisión de la voz. Para este caso, el códec que reúne las características citadas es el G.711, que fue descrito anteriormente en este capítulo, en la sección de VoIP.

2.3.7. Análisis de los sistemas de reconocimiento de voz open source

- **SPHINX-4**

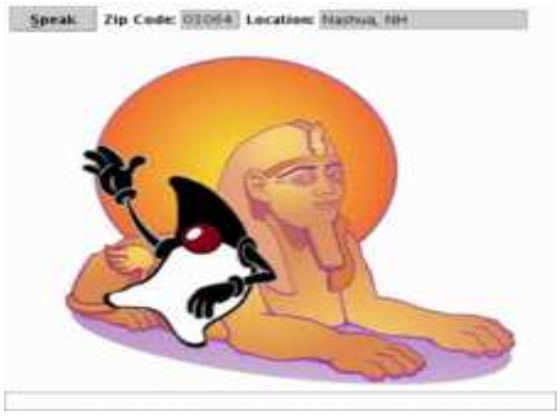

Figura 2.11: Logo de Sphinx-4 [10]

Sphinx-4 [10] es un proyecto desarrollado por el grupo CMU Sphinx en colaboración con Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs, y Hewlett Packard, con contribución de University of California at Santa Cruz y Massachusetts Institute of Technology (MIT). Entre sus características destacan:

1. Fue diseñado con el objetivo de permitir el control de aplicaciones mediante comandos de voz. Actualmente, se está trabajando en mejorar sus prestaciones para adaptarlo a aplicaciones de dictado. Aún así, su tasa de error sigue siendo bastante elevada al aplicarlo en este tipo de situaciones.

2. En la actualidad, es el proyecto que presenta un mayor nivel de desarrollo, en cuanto a la publicación y mejora de modelos acústicos y de lenguaje para dar soporte a distintos idiomas, en comparación con los demás sistemas de reconocimiento de voz. Esto es muy importante, ya que es señal de que saldrán nuevas versiones mejoradas.

3. Sin lugar a duda, es el mejor documentado, facilitando así el trabajo de los desarrolladores.

4. Está desarrollado completamente en Java. Desde el grupo CMU aseguran que ello no implica un rendimiento reducido, ya que, por ejemplo, la versión anterior de Sphinx (Sphinx-3) estaba desarrollada en C y no por ello ofrece un rendimiento superior.

5. Reconoce señales tanto continuas (locución continua) como discretas (frase a frase).

6. Dispone de distintas capacidades de reconocimiento, en cuanto al tamaño del vocabulario se refiere. En nuestro caso nos interesa utilizar el vocabulario más grande posible. El vocabulario más grande que posee Sphinx-4 se conoce con el nombre de

HUB4. Es un vocabulario de aproximadamente 64000 palabras (en inglés) y, al aplicarlo, se obtiene una tasa de error (teórica) de en torno al 20%.

7. La versión Sphinx-4 no está disponible para su uso en dispositivos móviles. Sin embargo, el grupo CMU ofrece versiones distintas, como PocketSphinx, para dicho propósito.

8. Como la mayoría de sistemas de reconocimiento de voz, se apoya en los Modelos Ocultos de Markov para llevar a cabo el proceso de reconocimiento.

9. Diseñado principalmente para su uso en sistemas Linux/Unix, aunque también puede usarse en Windows con ayuda del entorno *cygwin*. Además ha sido probado en sistemas operativos como Solaris y MAC OS X.

Conclusión

Como característica decisiva y diferenciadora frente al resto de sistemas, señalar que es el único que dispone de modelos acústicos adaptados a conversaciones telefónicas (incluido VoIP).

- **JULIUS**

Figura 2.12: Logo de Julius [11]

Julius [11] es un sistema de reconocimiento de voz en tiempo continuo, que viene siendo desarrollado por investigadores del Universidad de Kyoto (Japón) desde 1997 hasta la actualidad. Entre sus características destacan:

1. Alta precisión en la tarea de reconocimiento.

2. Su finalidad es la de permitir reconocimiento de voz como aplicación de dictado, es decir, soportando extensos vocabularios y gramáticas complejas.

3. Lenguaje de programación utilizado: C

4. Diseñado principalmente para su uso en sistemas Linux/Unix, aunque también puede usarse en Windows con ayuda del entorno *cygwin/mingw*.

5. Se distribuye con licencia gratuita junto con el código fuente.

6. Permite reconocimiento en tiempo real.

7. Su arquitectura le otorga una alta velocidad de procesamiento y un bajo consumo de recursos.

8. Altamente configurable. Permite ajustarse a los requerimientos de cada situación utilizando los algoritmos más adecuados para cada caso.

9. Documentado en japonés. Se está trabajando en la traducción de un manual al inglés, aunque no es tan completo como el original. Por lo tanto, en lo que se refiere a documentación no es muy recomendable.

10. También permite reconocimiento de comandos mediante su versión llamada Julian.

Conclusión

Hasta este punto, Julius es sin duda el motor de reconocimiento de voz más potente de todos. Pero presenta un grave inconveniente que es la falta de modelos acústicos de calidad. Y es que, la calidad del modelo acústico influye de manera decisiva en los resultados.

Posee un modelo acústico y de lenguaje para el japonés muy completo, y se está trabajando en el desarrollo de modelos para el inglés, pero aún son bastante rudimentarios. Para el resto de idiomas no dispone de ningún tipo de soporte.

- **HTK**

Figura 2.13: Logo de HTK [12]

HTK [12] son las siglas de Hidden Markov Model Toolkit. Fue desarrollado en el departamento de ingeniería de la Universidad de Cambridge en 1989 por Steve Young. Entre sus características destacan:

1. Como su propio nombre indica, no es un sistema de reconocimiento de voz en sí mismo, sino una herramienta para la creación y manipulación de Modelos Ocultos de Markov. Es decir, permite, entre otras cosas, crear los modelos acústicos necesarios para llevar a cabo el reconocimiento de voz, por parte de un decodificador (reconocedor de voz). Por ello, suele usarse en conjunto con un sistema de reconocimiento de voz independiente (por ejemplo, suele usarse junto a Julius). No obstante, sus desarrolladores, también han diseñado sus propios sistemas de reconocimiento de voz (decodificadores). En concreto, han desarrollado dos: HVite, pensado para aplicar reconocimiento de voz sobre vocabularios pequeños; y HDecode, utilizado para hacer reconocimiento de voz sobre grandes vocabularios.

2. Consiste en un conjunto de librerías desarrolladas en C.

3. Microsoft posee actualmente la licencia de HTK. De este modo, permite construir aplicaciones usando HTK, pero presenta restricciones en su distribución.

4. Puede ser ejecutado en los principales sistemas operativos: Windows, Linux/Unix, MAC OS X.

5. En lo que se refiere a HDecode, está diseñado, al igual que Julius, para reconocer grandes vocabularios. Por lo tanto, podría ser una buena opción, si no fuese porque presenta el mismo inconveniente que la mayoría, y es la ausencia de modelos acústicos, ya entrenados, para cubrir grandes vocabularios.

- **ISIP**

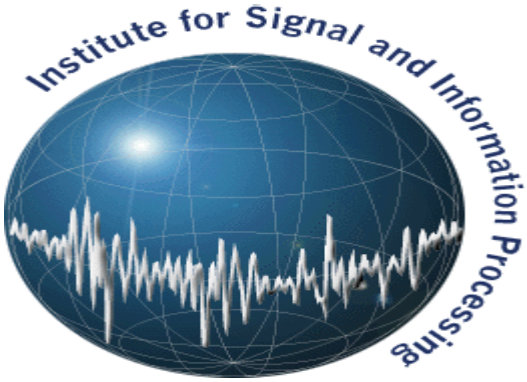

Figura 2.14: Logo de ISIP [13]

ISIP [13] son las siglas de Institute for Signal and Information Processing. Este proyecto se inició en 1994. Centra sus objetivos en el desarrollo de tecnologías del lenguaje humano para aplicaciones de inteligentes. Entre sus características destacan:

1. Es, como la mayoría, un sistema con capacidad limitada, que le permite soportar únicamente pequeños vocabularios. No es útil, por consiguiente, para aplicaciones de dictado.

2. No dispone de modelos acústicos y de lenguaje propios. Está pensado para que el usuario lo instale en su PC y cree su propio modelo acústico y de lenguaje adaptado a sus necesidades.

3. Desarrollado en C++.

4. Utiliza la herramienta HTK como soporte para algunas de sus aplicaciones.

5. Ofrece escasa información en lo que se refiere a características del sistema. Sin embargo, presenta una amplia documentación sobre el proceso de desarrollo de la aplicación.

- **SIMON**

Simon [14] es un proyecto a nivel europeo cuya finalidad consiste en proporcionar herramientas útiles a personas con minusvalías a la hora de usar ciertas aplicaciones como chat, mail, etc.

Simon es, básicamente, un interfaz gráfico que se ejecuta sobre HTK.

Aspectos a destacar:

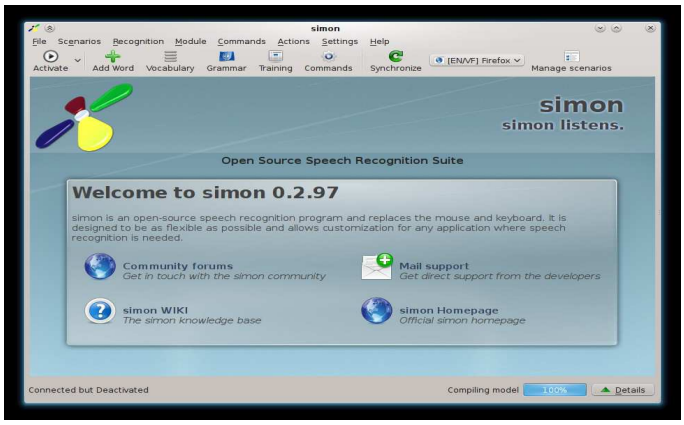

Figura 2.15: Página de inicio de Simon [14]

1. No provee modelos acústicos y de lenguaje ya que su propósito es permitir al usuario crear sus propios modelos desde cero. Es, por lo tanto, un programa diseñado para el control de aplicaciones mediante comandos, y no para dictado.

2. No permite su uso como librería para el desarrollo de nuevas aplicaciones. Por lo tanto, no es apto para nuestros fines.

3. Su arquitectura es bastante distinta a las de otros sistemas de reconocimiento de voz.

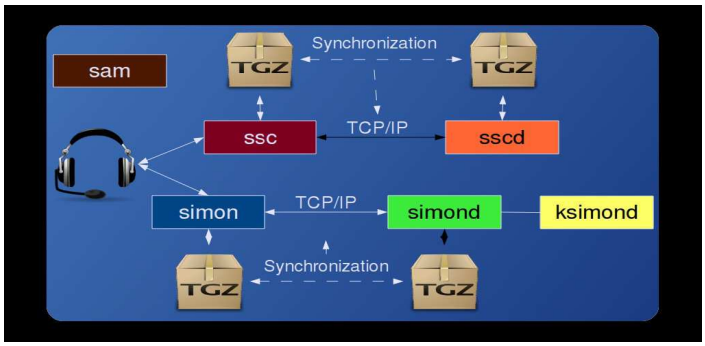

Figura 2.16: Arquitectura de Simon [14]

- **MARF**

MARF [15] son las siglas de Modular Audio Recognition Framework, y es un sistema de reconocimiento de voz desarrollado por la Universidad Concordia de Canadá, entre 2002 y 2008. Características:

1. Como limitación principal, señalar que su propósito es el de ofrecer un sistema de reconocimiento de voz para uso particular, esto es, con capacidad limitada a vocabularios pequeños. Por lo tanto, no permite dictado.

2. Tampoco dispone de modelos acústicos y de lenguaje, ya que la idea es que el usuario los cree adaptándolos a sus necesidades. Tan sólo contiene una colección de algoritmos diseñados para el procesamiento de lenguaje.

3. Está desarrollado en Java. Argumentan que esta decisión se tomó debido a las ventajas que este lenguaje aporta en cuanto a la portabilidad y gestión de memoria, de modo que les permitiese concentrarse plenamente en el desarrollo de los algoritmos.

4. Se distribuye bajo licencia BSD y está disponible para sistemas Unix y Windows (usando *cygwin*).

5. Dispone de un manual muy completo en el que se exponen todas las características del software, así como el proceso de instalación, resultados previstos, etc.

- **openSMILE**

openSMILE [16] es un sistema de reconocimiento de audio, especializado en la voz humana y el audio musical. Se le conoce también como "*the Munich open Speech and Music Interpretation by Large Space Extraction toolkit*" y fue desarrollado en la Universidad Técnica de Munich en el año 2010. Características principales:

1. Una vez más, la principal limitación de este software es su diseño, que lo capacita para el uso personal pero no para aplicaciones de dictado. Además, la ausencia de modelos acústicos y de lenguaje adaptados a dicho sistema, limitan aún más su utilidad para el presente proyecto.

2. Está desarrollado totalmente en C++ y cuenta con una arquitectura eficiente y flexible. Puede ejecutarse en los principales sistemas operativos: Linux, Windows y MacOS.

3. Fue diseñado principalmente para el reconocimiento de audio en tiempo real, pero también dispone de la capacidad de reconocimiento en modo off-line, cuanto se necesita procesar grandes cantidades de datos.

4. Posee una doble licencia. Una es GPL, destinada a todos aquellos desarrolladores que quieran usar el software libremente y contribuir a su mejora. La otra es especial, y está destinada a aquellas empresas que quieran utilizar el software pero no compartir su código.

5. Al igual que MARF, posee un manual muy completo como guía para desarrolladores.

6. Se apoya en la herramienta HTK y utiliza Modelos Ocultos de Markov como algoritmo para el procesado de señales.

- **SPRAAK**

Figura 2.17: Logo de SPRAAK [17]

SPRAAK [17] son las siglas de "*Speech Processing, Recognition and Automatic Annotation Kit*". Además, la palabra *spraak* significa "hablar" en alemán.

Es un proyecto derivado de otro que comenzó hace más de 15 años y se centraba en la investigación en Modelos Ocultos de Marvok (HMM). Fue llevado a cabo por el ESAT Speech Group, que a su vez es un subgrupo del Center for Processing Speech and Images (PSI), perteneciente al Departement of Electrical Engineering of the K.U.Leuven, Bélgica.

Aspectos a destacar:

1. Tampoco es útil para el proyecto, pues es una herramienta diseñada para proporcionar a usuarios particulares la funcionalidad de reconocimiento de voz en su PC.

2. En cuanto a lo que se refiere a modelos acústicos y de lenguaje, este proyecto sólo ofrece soporte para alemán.

3. Es una herramienta flexible y eficiente que cuenta con un decodificador basado en HMM.

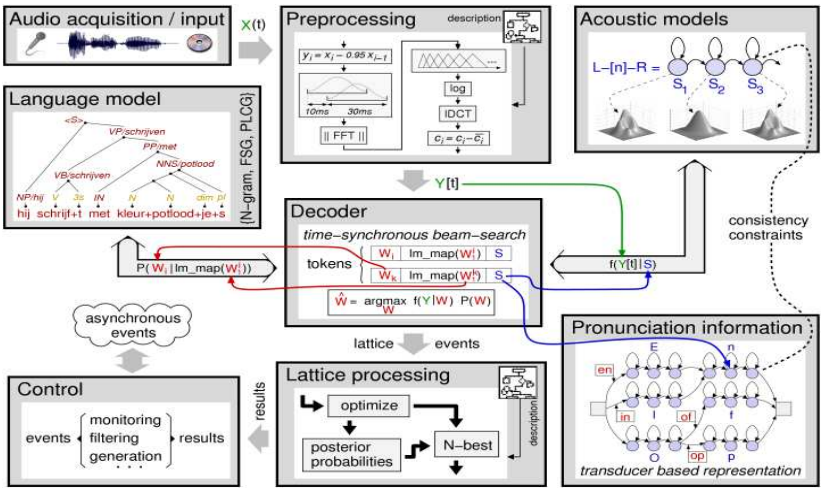

Figura 2.18: Arquitectura de SPRAAK [17]

4. La documentación ofrecida es muy completa, incluyendo desde códigos de ejemplo, hasta fórmulas matemáticas que describen la lógica interna del sistema.

5. Los módulos que integran el sistema están desarrollados en C. También usa Python como lenguaje de *scripting*. Puede ejecutarse en plataformas Unix/Linux y Windows.

- **iATROS**

iAtros [18] son las siglas de Automatic Trainable RecOgnition System. Es un sistema de reconocimiento de voz desarrollado por ingenieros del Instituto Tecnológico de Informática de la Universidad Politécnica de Valencia. Características principales:

1. Presenta como novedad respecto a los demás, el hecho de que, además del reconocimiento de voz, permite reconocimiento de escritura. Por lo tanto, su principal característica es su multifuncionalidad. Pero, como la mayoría de sistemas, decepciona en lo que se refiere al propósito del presente proyecto, que es el reconocimiento de voz aplicado a conversaciones (aplicación de dictado) sobre VoIP.

2. Está desarrollado en C y utiliza algoritmos de Modelos Ocultos de Markov.

3. Tampoco dispone de modelos acústicos y de lenguaje propios, por lo que el usuario debe crearlos desde cero, y adaptados a su voz.

4. Ofrece bastante documentación técnica, pero muy poca respecto a las características internas del sistema.

2.3.8. Conclusión

Para realizar este estudio sobre los distintos sistemas de reconocimiento de voz, se ha seguido el siguiente proceso:

1. Buscar información sobre los sistemas de reconocimiento de voz disponibles.

2. Como resultado se obtuvo un gran número de opciones. Se continuó la búsqueda descartando aquellos que no eran proyectos de software libre.

3. Como consecuencia, la lista se redujo drásticamente, quedando como resultado únicamente aquellos sistemas que se han citado con anterioridad.

4. Posteriormente, se consultaron foros que ofrecían opiniones sobre estos sistemas. Y todos coincidían en que Julius y Sphinx-4 eran los sistemas mejor capacitados.

5. Julius destaca sobre todo por su precisión y rapidez. Pero la falta de modelos acústicos y de lenguaje adaptados a él lo descartan. Y como él, el resto de sistemas presentan la misma limitación. Por lo tanto quedan descartados, ya que para poder alcanzar un rendimiento similar al que ofrece Sphinx-4 se requerirían cientos de horas de entrenamiento.

6. El tema relacionado con los modelos acústicos y de lenguaje se presenta como el primer aspecto a tener en cuenta a la hora de elegir un sistema de reconocimiento de voz. Esto se debe a que la elaboración de éstos es una tarea de larga duración. Por ello, disponer de una larga lista de modelos bien elaborados y adaptados a diferentes idiomas, voces y canales es una enorme ventaja.

7. Sphinx-4 destaca por ser el sistema que da soporte a un mayor número de idiomas, gracias a la gran cantidad de modelos acústicos y de lenguaje que ofrece, así como la calidad de estos. Más aún, es el único que dispone de modelos acústicos entrenados para conversaciones telefónicas o de VoIP. Además, quizá no sea el sistema con mayor precisión y rapidez, pero es el más popular y el que ofrece mayores facilidades a los desarrolladores en la actualidad.

8. Además, la eficiencia en los sistemas de reconocimiento de voz es algo difícil de medir y comparar. Al fin y al cabo, sus arquitecturas son muy similares. Todos se basan en los Modelos Ocultos de Markov y utilizan las herramientas más avanzadas en reconocimiento de voz, tales como: "*Feature Extraction of Cepstra, FFT, Mel Frecuency Filters*, etc".

9. Lo cual lleva a deducir que la eficiencia de un sistema de reconocimiento de voz no la determina tanto su arquitectura como la calidad de los modelos acústicos y de lenguaje con los que ha sido entrenado. Buena muestra de ello es el hecho de que Julius ofrece un rendimiento muy inferior al de Sphinx-4 a la hora de transcribir audio en inglés.

10. Con todo lo anterior, se decidió elegir Sphinx-4 como sistema de reconocimiento de voz para este proyecto.

11. Como apunte final señalar que, ya que el proceso de transcripción no es perfecto, se use el sistema que se use, es necesario corregir las transcripciones resultantes.

CAPÍTULO 3: DISEÑO DEL SISTEMA

En este capítulo se describe la arquitectura y la funcionalidad global del sistema, así como de Sphinx-4. Se decidió incluir la parte funcionamiento y diseño de Sphinx-4, ya que es el componente más importante de la aplicación y merece una descripción detallada.

La siguiente figura muestra la arquitectura general del sistema:

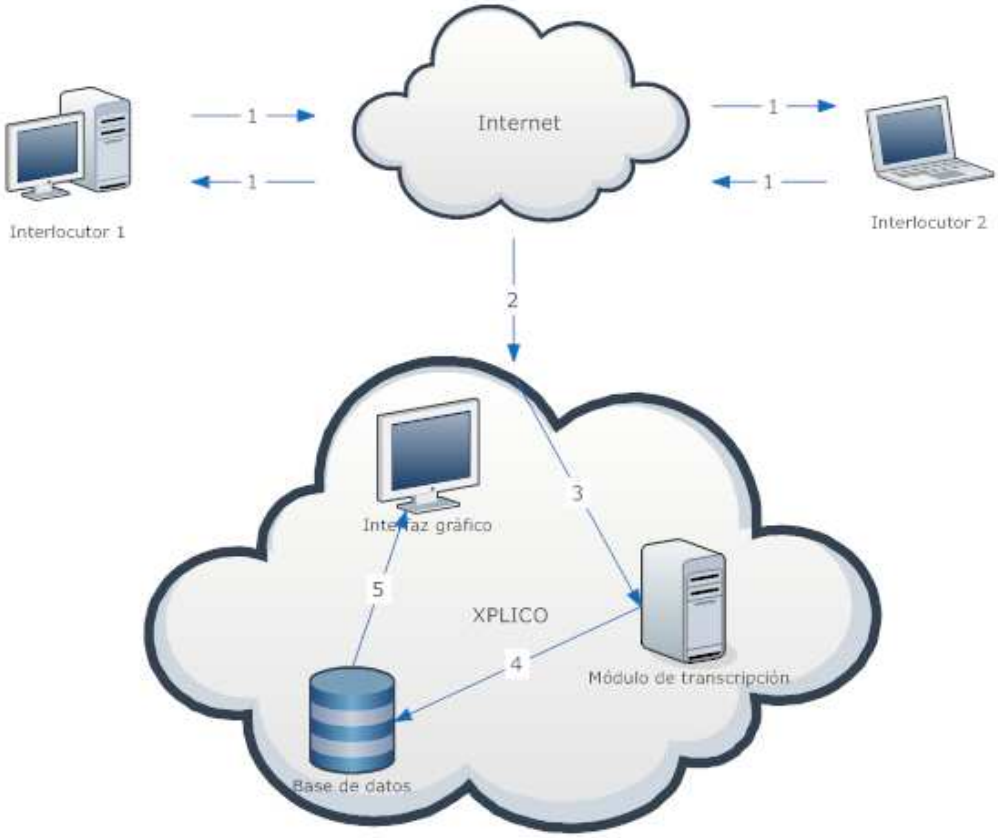

Figura 3.1: Arquitectura global del sistema

A modo de introducción, se describe la funcionalidad general del sistema:

1. En primer lugar, se tiene el escenario en el que dos individuos se comunican a través de Internet, utilizando VoIP. Estos dos individuos son sospechosos de cometer alguna actividad delictiva, y están siendo monitorizados por orden judicial.

2. Para llevar a cabo dicha monitorización, la policía utiliza la plataforma de interceptación legal de comunicaciones desarrollada por el proyecto INDECT, la cual se basa en el software de análisis forense Xplico. Con este software se pretende interceptar y analizar todos los contenidos capturados. De entre todos los contenidos interceptados, los que se pretenden analizar en este proyecto son aquellos relacionados con conversaciones realizadas mediante VoIP.

3. Para ello, Xplico entregará al módulo de transcripción los dos flujos RTP de audio que componen la conversación, a saber, uno por cada interlocutor. Este módulo procederá a procesar los dos ficheros de audio, generando una transcripción en texto por cada uno de ellos. Posteriormente, ambas transcripciones (la de cada locutor por separado) se unirán en una sola y se ordenará por marcas de tiempo, para mantener el orden temporal de la conversación.

4. Cuando se disponga de la transcripción de la conversación completa, se actualizará la base de datos de Xplico, creando una nueva entrada para referenciar a la nueva conversación, así como a los elementos asociados a ella (ficheros de audio y transcripción).

5. Por último, cuando el analista de la policía desee analizar el contenido de la conversación, se le mostrará mediante un interfaz gráfico, todos los contenidos relacionados con ésta, de manera estructurada y accesible, para facilitarle, en la medida de lo posible, su labor de análisis.

3.1. ARQUITECTURA GLOBAL DEL SISTEMA

En esta sección se describen los módulos que integran la aplicación así como su interrelación. Básicamente, se podría decir que el sistema está formado por dos módulos principales y una lógica de procesamiento que conecta ambos.

3.1.1. Módulo Xplico

El software de Xplico, como núcleo de la plataforma de interceptación legal de comunicaciones, se comunicará, primero con el módulo de transcripción, y después con el usuario, todo ello mediante una arquitectura Modelo-Vista-Controlador. Más concretamente, será el encargado de:

- Proporcionar los datos necesarios al módulo de transcripción. Básicamente, estos datos serán los ficheros de audio de cada interlocutor.

- Almacenar en su base de datos los resultados obtenidos por el módulo de transcripción, es decir, agrupar los ficheros de audio de cada interlocutor, el fichero de audio de la conversación completa y la transcripción de la conversación en un único bloque denominado *Conversation*. Cada fichero que integra un *Conversation* se identificará como un *Stream*. De ese modo, cada *Conversation* estará formado por cuatro *Streams* (tres ficheros de audio y la transcripción de la conversación).

- Interactuar con el usuario a través del interfaz gráfico, permitiéndole acceder a los ficheros generados con anterioridad, así como editarlos y entrenar el sistema con los mismos.

3.1.2. Módulo de transcripción

Su principal componente es Sphinx-4. Sus funciones principales son:

- Convertir el formato de los ficheros de audio de cada interlocutor que recibe a través de Xplico, de .mp3 a .wav.

- Realizar la transcripción del fichero de audio .wav de cada interlocutor por separado.

- Mezclar las dos transcripciones resultantes en una sola, ordenada por marcas de tiempo (*timestamps*).

3.1.3. Lógica de procesamiento

No es un módulo en si, pero realiza una labor fundamental para la correcta ejecución de la aplicación. Se podría decir que es el cerebro de la aplicación, la parte que se encarga de interconectar cada módulo con los demás. Consiste en un conjunto de scripts de Bash que ejecutan tareas como:

- Llamadas a Sphinx-4 y a la base de datos de Xplico.

- Procesamiento de los ficheros de transcripción.

- Creación de los directorios en los que se almacenarán las conversaciones.

- Manipulación de audio.

- Gestión del proceso de entrenamiento de locutores.

- Procesamiento de llamadas desde el interfaz gráfico de Xplico.

- Etc.

3.2. FUNCIONALIDAD GLOBAL DEL SISTEMA

En lo que se refiere al esquema de funcionamiento global de la aplicación, éste se podría dividir en dos vertientes: Proceso automático de transcripción y proceso dinámico de interactuación con el usuario.

3.2.1. Proceso automático de transcripción

Es aquel en el que no interviene el usuario. Se ejecuta automáticamente en cuanto se interceptan los flujos VoIP.

A continuación se muestra el diagrama de flujo que representa dicho proceso:

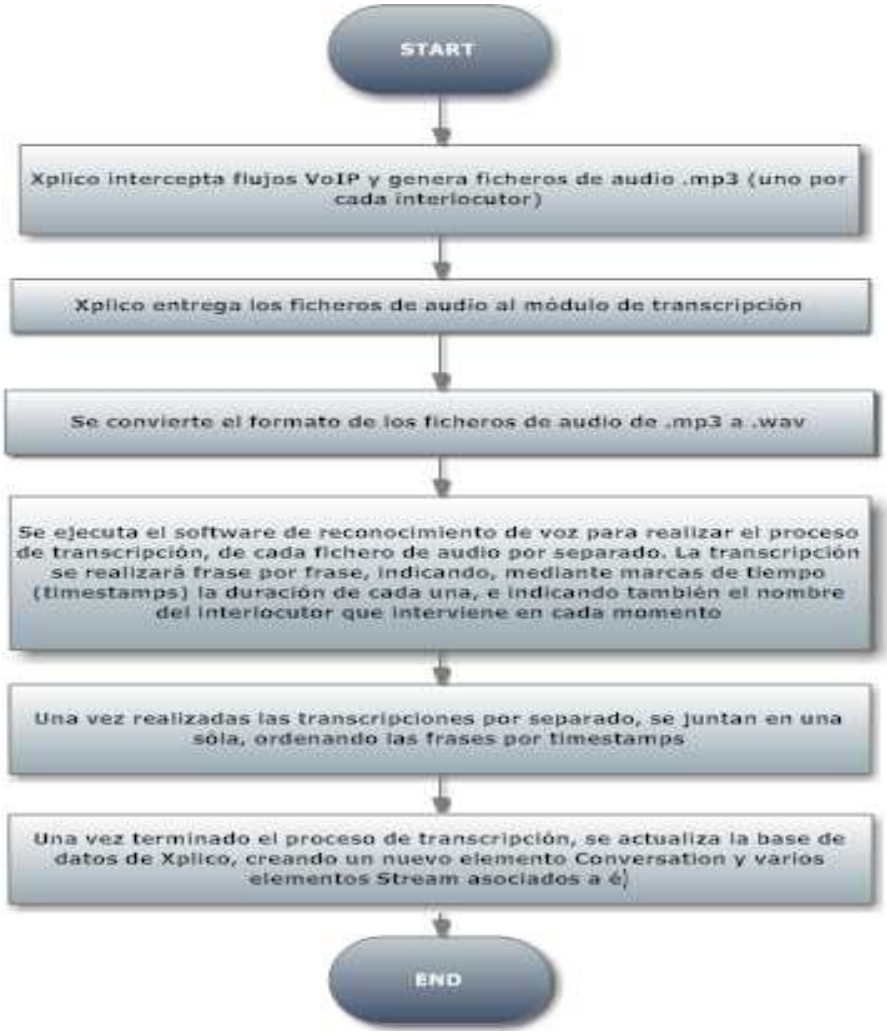

Figura 3.2: Proceso automático de transcripción

3.2.2. Proceso dinámico de interactuación con el usuario

Es aquel en el que el sistema interactúa con el usuario mediante el Modelo-Vista-Controlador. Se iniciará cuando el usuario utilice el interfaz gráfico.

A continuación se muestra el diagrama de flujo que representa dicho proceso:

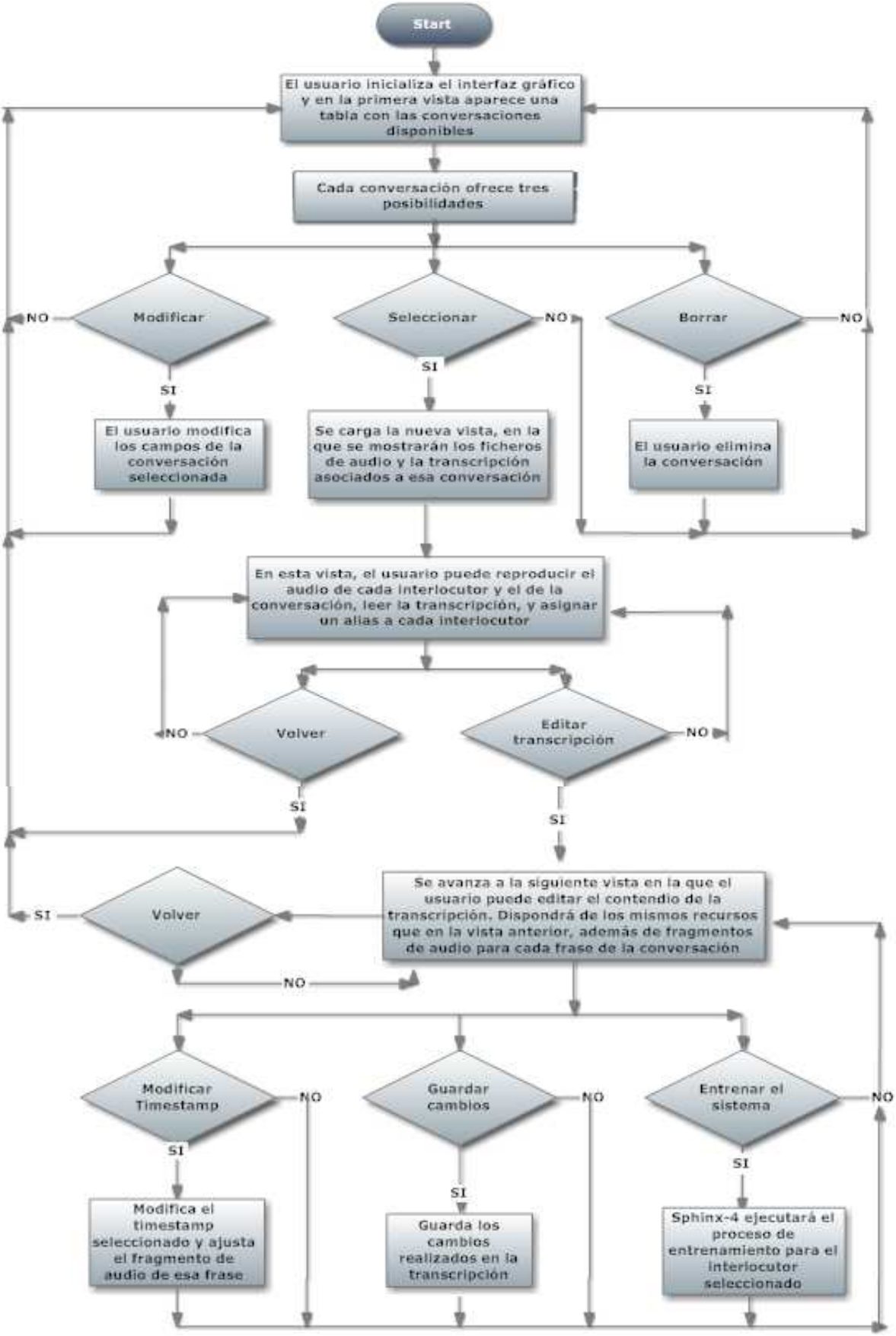

Figura 3.3: Proceso dinámico de interactuación con el usuario

En este punto es conviene aclarar dos aspectos del diagrama de flujo anterior:

- ¿Para qué sirve modificar los *timestamps*?
  Modificar los *timestamps*, que indican el comienzo y final de cada frase de la transcripción, no es obligatorio, pero es una herramienta útil que permite al analista ajustar la duración del fragmento de audio que corresponde con esa frase, y así poder comprobar con mayor facilidad si la transcripción generada por el software de reconocimiento de voz es correcta, o no, en cuyo caso, podrá corregirla más rápidamente.

- ¿En qué consiste el proceso de entrenamiento?
  El proceso de entrenamiento es otra funcionalidad opcional que permite ir mejorando la precisión del sistema de reconocimiento de voz, a medida que se le introducen nuevos datos (corregidos) sobre un interlocutor concreto, de manera que se vaya adaptando cada vez con mayor precisión a las características específicas de su voz. Actualmente, una parte del entrenamiento - más concretamente la generación del diccionario - se realiza mediante un servicio online ofrecido por el grupo CMU Sphinx. Obviamente, este procedimiento deberá sustituirse por otro que garantice la confidencialidad de los datos, cuando lo utilice la Policía.

3.3. ARQUITECTURA Y FUNCIONAMIENTO DE SPHINX-4

3.3.1. Funcionamiento

Como ya se explicó en el capítulo anterior, Sphinx-4 [10] es un sistema de reconocimiento de voz basado en Modelos Ocultos de Markov (HMM). En los sistemas basados en este modelo estadístico, las unidades sonoras se denominan fonemas y se representan mediante un modelo estadístico que indica la distribución de todas las muestras que componen el fonema. Esto se conoce como modelo acústico del fonema.

Cuando se crea un modelo acústico, las señales de voz son primero transformadas en una secuencia de vectores que representan ciertas características de la señal. Después, los parámetros del modelo acústico son estimados usando esos vectores. Este proceso se conoce como entrenamiento de modelos acústicos.

Durante el reconocimiento de voz, los vectores se extraen de la señal de entrada, igual que en el proceso de entrenamiento. Luego, estos vectores extraídos en tiempo real se comparan con el modelo acústico. Los resultados obtenidos indican la probabilidad de que un conjunto de vectores concreto pertenezcan al fonema del correspondiente modelo acústico. El componente del reconocedor (*Recognizer*) que se encarga de generar estos vectores es el interfaz de entrada (*FrontEnd*).

El proceso de reconocimiento de voz consiste, por tanto, en obtener la mejor secuencia de palabras posibles que coincida con la señal de entrada. Es, básicamente, un problema de búsqueda y, en el caso de los sistemas basados en Modelos Ocultos de Markov (HMM), un problema de búsqueda en grafos. Un grafo está compuesto por fonemas concatenados siguiendo un orden concreto, tal y como especifique la gramática del sistema. Como ejemplo, en la siguiente figura se muestra como un grafo de búsqueda que decodifica las palabras “one” y “two”:

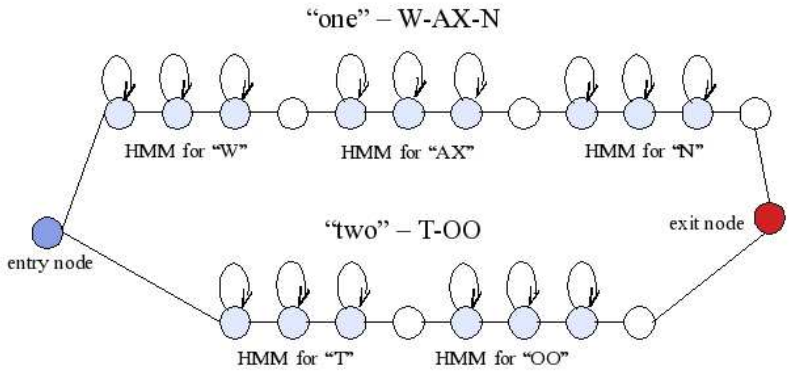

Figura 3.4: Ejemplo de HMM [10]

Construir el grafo anterior requiere la participación de varios componentes: Primero, requiere un diccionario que descompone la palabra "one" en los fonemas W, AX y N, y la palabra "two" en T y OO. También requiere el modelo acústico que obtenga el HMM de los fonemas W, AX, N, T y OO. En Sphinx-4, la labor de construir el grafo de búsqueda es llevada a cabo por el módulo de lingüística.

Normalmente, el grafo de búsqueda también contiene información sobre la probabilidad de ocurrencia de ciertas palabras. Esta información la proporciona el modelo de lenguaje. Supongamos que, en el ejemplo anterior, la probabilidad de que alguien diga "one" (EJ: 0.8) es mucho mayor que la de decir "two" (0.2). Entonces, en el grafo anterior, la probabilidad de transición entre el nodo de entrada y el primer nodo del HMM para la W será de 0.8, mientras que la probabilidad de transición entre el nodo de entrada y el primer nodo del HMM para T será 0.2. El camino hacia "one" tendrá, consecuentemente, un pesomayor.

Una vez que el grafo está construido por completo, se compara la secuencia de vectores con las diferentes rutas que componen el grafo, hasta encontrar la que más se parezca. La mejor ruta suele ser la que tiene menor coste o mayor puntuación, dependiendo de la implementación. En Sphinx-4, el proceso de búsqueda a través del grafo para encontrar el mejor camino lo realiza el gestor de búsqueda (*Search Manager*).

Como se puede observar en el grafo anterior, muchos nodos tienen transiciones propias. Esto puede originar un gran número de posibles caminos a través del grafo. Como resultado, encontrar la mejor ruta posible puede llevar mucho tiempo. El objetivo del simplificador (*Pruner*) es reducir el número de caminos posibles durante la búsqueda, utilizando técnicas heurísticas como eliminar los caminos con puntuaciones menores.

Resumiendo, la señal de entrada se transforma en una secuencia de vectores que contienen las características acústicas de dicha señal. Una vez todos los vectores hayan sido decodificados, se buscan todos los caminos que hayan alcanzado el nodo de salida. Se selecciona el camino con la puntuación más alta es el seleccionado y se genera un resultado con todas las palabras (o fonemas) que forman ese camino.

3.3.2. Arquitectura

La siguiente figura muestra la arquitectura de Sphinx-4 y los principales componentes que la integran:

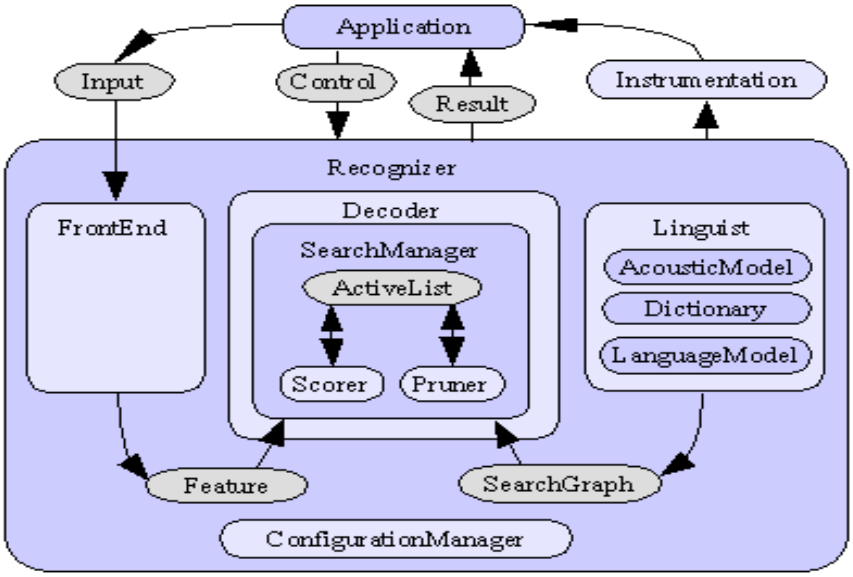

Figura 3.5: Arquitectura y principales componentes de Sphinx-4 [10]

Cuando el Reconocedor arranca, inicia el interfaz de entrada (el cual genera vectores de características acústicas a partir de la señal de entrada), el decodificador y la lingüística (que genera el grafo de búsqueda), de acuerdo a la configuración especificada por el usuario. Estos tres componentes iniciarán, a su vez, sus correspondientes subcomponentes. La lingüística creará el modelo acústico, el diccionario y el modelo de lenguaje. A partir de ellos, utilizando la información que contienen, creará el grafo de búsqueda. El decodificador iniciará el gestor de búsqueda, el cual, a su vez, creará el clasificador, simplificador y la lista activa.

La mayoría de estos componentes se modelan con interfaces Java. Como interfaces que son, pueden tener diferentes implementaciones. Para saber cual es la implementación que debe utilizar el sistema, ésta se especifica en un fichero de configuración XML, controlado por el gestor de configuraciones (*Configuration Manager*). En este fichero de configuración, el usuario también puede especificar las propiedades que posee la implementación. Una de esas propiedades puede ser, por ejemplo, la tasa de muestreo de la señal de voz de entrada.

El gestor de configuraciones tiene, por tanto, dos funciones principales:

1. Determinar que componentes van a ser usados por el sistema. Sphinx-4 es un sistema altamente flexible que permite reemplazar y/o modificar cualquier componente en tiempo de ejecución.

2. Especificar detalladamente la configuración de cada uno de esos componentes.

La estructura del fichero de configuración es la siguiente:

- Nombre y tipos de todos los componentes del sistema
- Conectividad de los componentes, es decir, como se comunican unos con otros.
- Configuración detallada de los componentes.

Las siguientes líneas son un ejemplo de la configuración detallada de un componente:

```
<config>

    <component name="concatDataSource"
type="edu.cmu.sphinx.frontend.util.ConcatFileDataSource">
        <property name="sampleRate" value="16000"/>
        <property name="transcriptFile" value="reference.txt"/>
        <property name="silenceFile"
value="/lab/speech/sphinx4/data/tidigits/test/raw16k/silence1sec.ra
w"/>
        <property name="bytesPerRead" value="320"/>
        <property name="batchFile" value="tidigits.batch"/>
        <property name="addRandomSilence" value="true"/>
    </component>

</config> >
```

3.3.3. Componentes

Sphinx-4 consta de tres módulos principales: Interfaz de entrada (*FrontEnd*), decodificador (*Decoder*) y lingüística (*Linguist*).

3.3.3.1.1. FrontEnd

Acepta como señal de entrada tanto ficheros de audio (.wav) como voz en tiempo real a través de micrófono.

Este interfaz contiene una o más cadenas paralelas compuestas procesadores de datos independientes, encargados de parametrizar la señal de entrada. El hecho de poseer varias cadenas trabajando en paralelo permite procesar diferentes señales simultáneamente.

Cada procesador recibe una señal de entrada y genera otra de salida, que será de entrada para el siguiente procesador. El último procesador es el encargado de agrupar todos los vectores en un solo objeto (*Data object*) y entregárselo al decodificador.

Habitualmente, Sphinx-4 se configura con un interfaz de entrada que produce *Mel frecuency cepstral coefficients (MFCCs).*

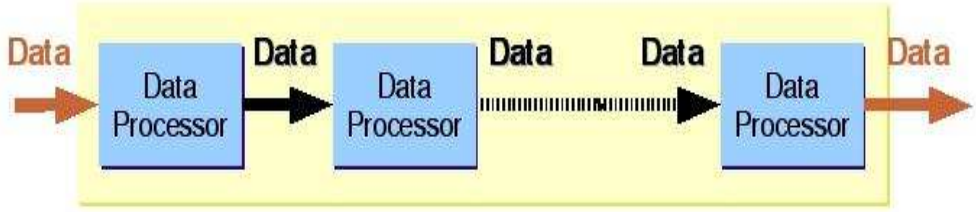

Figura 3.6: FrontEnd de Sphinx-4 [10]

3.3.3.2. Decoder

La misión principal del decodificador es utilizar los vectores provenientes del interfaz de entrada junto con el grafo de búsqueda generado por la lingüística para generar como resultado diferentes hipótesis.

Está compuesto por un gestor de búsqueda (*SearchManager)*, que es el componente más importante del decodificador, y otras aplicaciones de soporte que simplifican el proceso de decodificación a la aplicación principal.

El decodificador simplemente le dice al gestor de búsqueda que reconozca un conjunto de tramas que contienen vectores. En cada paso del proceso, el gestor de búsqueda crea un objeto Resultado que contiene todos los caminos que han alcanzado el estado final.

Un nodo (*token*) es un objeto asociado a un estado de búsqueda (*SearchState*) y contiene:

- Las puntuaciones lingüísticas y acústicas promediadas del camino, en un punto dado.
- Una referencia al estado de búsqueda, la cual permite al gestor de búsqueda relacionar el nodo con su estado de salida, los fonemas interdependientes, pronunciación y estado gramatical.
- Una referencia a la trama que contiene los vectores de entrada.

Una implementación típica del gestor de búsqueda consiste en agrupar un conjunto de nodos en forma de lista activa (*ActiveList*) en cada paso del proceso

La implementación del simplificador (*Pruner*) se facilita mucho gracias al recolector de basura de Java. Gracias a él, el simplificador puede determinar un camino completo simplemente eliminando el nodo sobrante del camino, desde la lista activa. El proceso de eliminar el nodo sobrante consiste en marcar dicho nodo y cualquier otro nodo que quede aislado del camino como inutilizables, permitiendo al recolector de basura liberar la memoria que ocupaban.

El gestor de búsqueda también se comunica con el clasificador (*Scorer*), un módulo de estimación de probabilidades de estados, que provee como salida los valores de densidad de estados demandados. Cuando el gestor de búsqueda solicita una puntuación para un estado en un determinado momento, el clasificador accede al vector de características acústicas y realiza cálculos matemáticos para obtener la puntuación.

El clasificador mantiene toda la información perteneciente a las salidas de densidades de estado. Esto se debe a que el gestor de búsqueda necesita saber si el proceso de puntuación se ha llevado a cabo con HMMs continuos, discretos o semi-continuos. Además, la función de densidad de probabilidad de cada estado HMM es aislada de la misma manera.

Cualquier algoritmo heurístico incorporado al procedimiento de puntuación, con el propósito de acelerarlo, puede ser ejecutado localmente junto al clasificador. Este último puede, además, mejorar su rendimiento si se dispone de varias CPUs para llevar a cabo el procesamiento.

En el caso de nuestra aplicación, la implementación del gestor de búsqueda que usaremos será "*WordPrunigBreadthSearchManager*", la cual realiza una búsqueda de tramas síncrona Viterbi (algoritmo que permite encontrar las secuencias de estados más probables en un HMM a partir de una observación) con un simplificador independiente que actúa sobre cada trama. En lugar de manejar una única lista activa, gestiona un conjunto de ellas, una por cada tipo de estados definidos por la lingüística. La simplificación se realiza descomponiendo en secuencias ordenadas los diferentes tipos de estados.

3.3.3.3. Linguist

La lingüística genera el grafo de búsqueda (*SearchGraph*) que utiliza el decodificador durante la búsqueda, encargándose al mismo tiempo de gestionar la complejidad involucrada en la generación de este grafo.

Es un módulo independiente, lo que facilita a los desarrolladores configurar el sistema con distintas implementaciones. Una implementación típica de la lingüística del sistema consiste en construir el grafo de búsqueda usando la estructura lingüística proporcionada por el modelo de lenguaje y la estructura topológica proporcionada por el modelo acústico (basado en HMM). También suele incorporar un diccionario, que contiene las reglas de pronunciación, para mapear las palabras contenidas en el modelo de lenguaje en secuencias de sonidos del modelo acústico.

Para el caso de nuestra aplicación usaremos la lingüística conocida como "*LexTreeLinguist*" que es la más apropiada para el reconocimiento de voz usando grandes diccionarios y modelos estocásticos *N-Gram*.

Resumiendo, la lingüística contiene tres elementos independientes, modelo de lenguaje, modelo acústico y diccionario, que serán descritos a continuación:

- ***Modelo de lenguaje***

  Este elemento provee una estructura léxica a la lingüística del sistema. Permite diversos tipos de implementaciones, que se pueden agrupar en dos categorías:

  - Gramáticas basadas en grafos: Es la implementación más simple y consiste en crear una lista limitada de palabras o fonemas con sus correspondientes grafos.

  - Modelos estocásticos *N-Gram*: Esta implementación es más compleja. Se utiliza para aplicaciones de dictado y consiste en secuencias de palabras y expresiones con sus distintas probabilidades.

En el caso concreto de nuestra aplicación usaremos un modelo estocástico *N-Gram* llamado "*LargeTrigramModel*", que es el modelo de lenguaje más extenso y potente proporcionado por Sphinx-4.

- ***Diccionario***

El diccionario provee las reglas de pronunciación para las palabras encontradas en el modelo de lenguaje. Estas reglas de pronunciación se construyen fragmentando las palabras en fonemas predefinidos por el modelo acústico.

En nuestro caso usaremos el diccionario más extenso de que dispone Sphinx-4 hasta la fecha, denominado "*cmudict.0.6d*".

- ***Modelo acústico***

El modelo acústico provee el mapeo entre fonemas y HMM, que puede ser comparado frente a los vectores provenientes del interfaz de entrada.

Generalmente, el módulo de lingüística descompone cada palabra del modelo de lenguaje en una secuencia de fonemas interdependientes. Estos fonemas son transferidos al modelo acústico preservando los grafos HMM asociados con ellos. A continuación, el modelo acústico utiliza esos grafos HMM junto con el modelo de lenguaje para construir el grafo de búsqueda.

A diferencia de la mayoría de los sistemas de reconocimiento, que representan los grafos HMM como una estructura fija en memoria, Sphinx-4 los representa directamente como un grafo de objetos, es decir simplifica el proceso. En este grafo, cada nodo se corresponde con un estado HMM y cada flecha representa la probabilidad de transición de un estado a otro del HMM.

Los componentes que conforman cada estado del HMM son combinaciones de gaussianas, matrices de transición, y combinaciones de pesos, entre otros.

Sphinx-4 ofrece, actualmente, una única implementación del Modelo Acústico (el componente) que permite cargar y utilizar modelos acústicos (versiones compatibles con el componente) generados por Sphinx-3 Trainer.

Hay gran variedad de modelos acústicos ofrecidos por el grupo CMU Sphinx, y en la apliación pueden usarse los que se deseen. Pero para la demostración y como modelo acústico principal se usará el "*US English Communicator Telephone Acoustic Model*", ya que está diseñado específicamente para conversaciones telefónicas (VoIP) en inglés.

CAPÍTULO 4: IMPLEMENTACIÓN DEL SISTEMA

En este capítulo se describe brevemente la implementación de los distintos componentes que integran la aplicación. Para seguir un orden determinado, primero se describirán los módulos relacionados con Sphinx-4. A continuación se hará lo propio con los módulos encargados de la lógica de procesamiento. Por último, se expondrán los módulos relacionados con el interfaz gráfico.

Antes de comenzar con el análisis individual de cada componente, es conveniente describir la estructura de directorios de la aplicación:

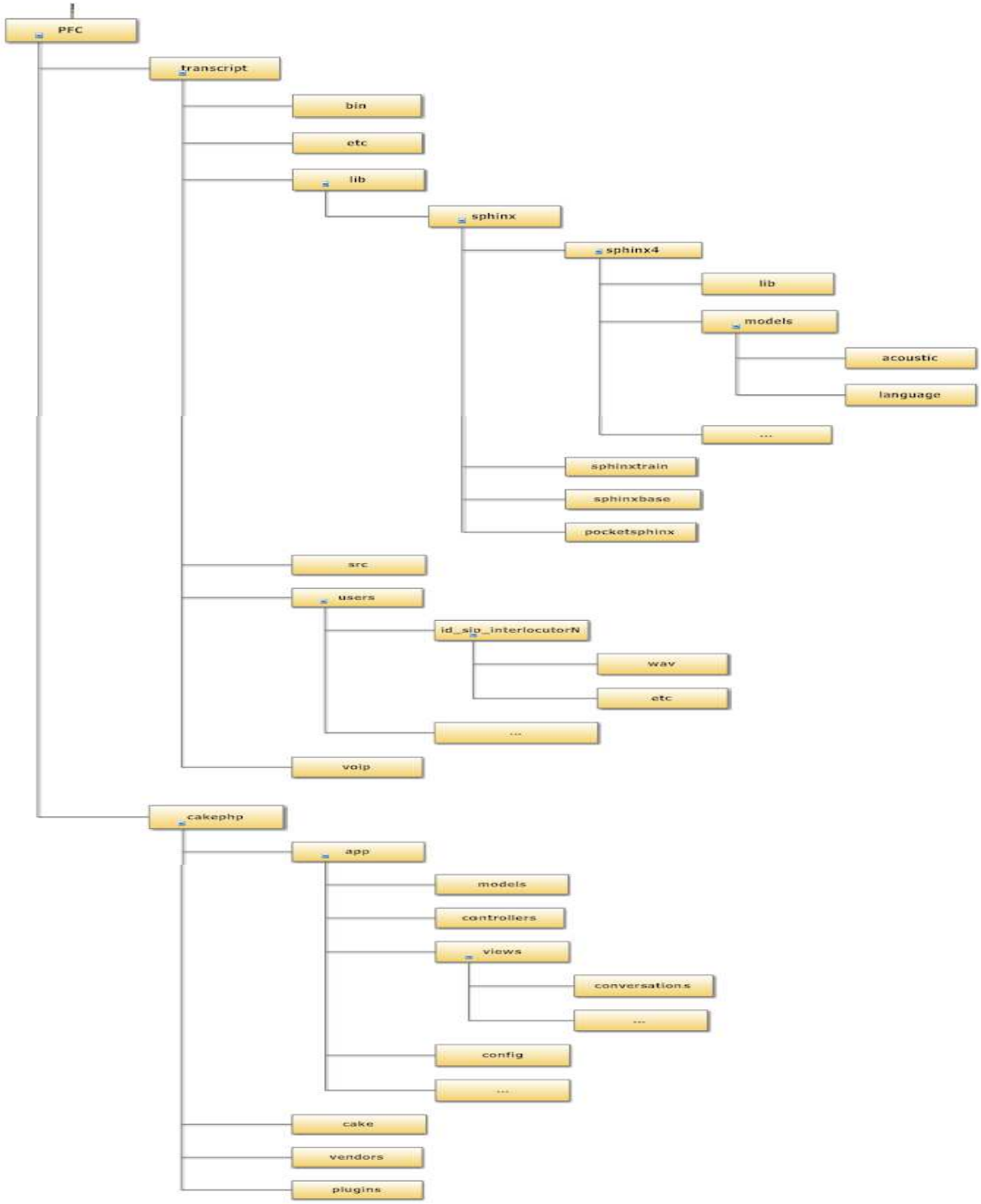

Figura 4.1: Estructura de directorios de la aplicación

Los componentes principales se encuentran en:

- ***/PFC/transcript/bin/***: Aquí se encuentran los scripts que integran la lógica de procesamiento, así como los ficheros *.class* de las clases Javas, y algunos ejecutables de Sphinx-4.

- ***/PFC/transcript/etc/:*** En esta ruta se encuentra el fichero de configuración de Sphinx-4.

- ***/PFC/transcript/src/:*** Contiene el programa Java que implementa la ejecución de Sphinx-4.

- ***/PFC/transcript/voip/:*** Aquí se almacenan las conversaciones (.wav) que Xplico entrega al módulo de transcripción, así como las transcripciones generadas a posteriori.

- ***/PFC/cakephp/app/controllers/:*** Contiene el código del Controlador de la aplicación PHP.

- ***/PFC/cakephp/app/views/concersations/:*** Contiene el código de las vistas de la aplicación PHP.

4.1. RECONOCIMIENTO DE VOZ

A continuación se describen los módulos encargados de la ejecución de Sphinx-4 para los fines específicos de este proyecto.

Lo primero de todo es mencionar que hay dos ficheros importantes: `Transcript.java` y `config.xml`.

- **`Transcript.java`**

Este programa Java es el que se encarga del proceso de transcripción. Su estructura es la siguiente:

1. Lo primero que hace es comprobar los parámetros que recibe. Deben ser, por este orden:
    - Identificador SIP del interlocutor.
    - Fichero de configuración (*config.xml*)
    - Fichero de audio del interlocutor.

2. Una vez recibidos correctamente los parámetros, lo siguiente que hace es crear el Gestor de Configuración (*ConfigurationManager*), a partir del fichero de configuración, `config.xml`.

3. Construye el Reconocedor (*Recognizer*) y le asigna los recursos disponibles (*Decoder, Linguist y FrontEnd*).

4. Configura la entrada de audio que utilizará el Reconocedor mediante la creación de un objeto *AudioFileDataSource*. A este objeto se le asocia el fichero de audio de entrada.

5. Una vez instanciados los objetos principales entra en un bucle, dentro del cual se ejecutarán instrucciones para procesar el audio, que finalizará cuando termine el fichero de audio.

6. Además de eso, deben referenciarse los *.jars* y el *.class* correspondientes. De eso se encarga el script `transcript.sh` (que será descrito más adelante), que es el encargado de ejecutar este programa *.java*.

- **`config.xml`**

Este fichero de configuración de Sphinx-4 contiene toda la información asociada a la configuración de los componentes que intervendrán para llevar a cabo del proceso de transcripción.

Dependiendo de la aplicación, pueden usarse distintos componentes y/o modificar los parámetros de cada uno de ellos. En este caso, se utilizó un fichero de configuración estándar de base, sobre el que se realizaron modificaciones y pruebas hasta obtener los resultados óptimos.

En el caso concreto de esta aplicación, intervienen los siguientes elementos:

- *Recognizer*: Es el motor que inicia/crea el resto de componentes.
- *Decoder:* Decodifica los vectores de características acústicas generados por el interfaz de entrada.
- *SearchManager*: Gestiona el grafo de búsqueda.
- *ActiveListManager:* Gestiona las listas de nodos de estados activos.
- *TrivialPruner:* Se encarga de simplificar los caminos entre nodos de estados.
- *ThreadedScorer:* Mantiene la puntuación de acústica de cada camino en distintos hilos.
- *LexTreeLinguist:* Este componente mantiene actualizado el estado del lenguaje (las probabilidades de cada fonema y frases).
- *AcousticModel*: ya que la aplicación consiste en transcribir audio transmitido a través de redes VoIP, usaré el modelo acústico más apropiado para este tipo de canal. Concretamente, el modelo acústico se llama *"US English Communicator Telephone Acoustic Model"*.
- *TrigramModel:* Este componente especifica el modelo de lenguaje que se utilizará. En el caso de la aplicación será *"US English HUB4 Language Model"*, que es el más apropiado para vocabularios extensos.
- *Dictionary:* Ahora mismo, todas aplicaciones de Sphinx-4 utilizan el mismo diccionario, *"cmudict.06d"*
- *UnitManager:* Gestiona las distintas unidades que integran el Reconocedor.
- *EpFrontEnd:* Este componente es el interfaz de entrada que, a su vez, posee una serie de subcomponentes. Entre ellos destacar el subcomponente llamado *"melFilterBank"*, que realiza labores de filtrado de frecuencias, y que deberá ser modificado para adaptarse a las frecuencias especiales del canal.

- *Monitors:* Como su propio nombre indica, este componente posee una serie de subcomponentes que nos permiten monitorizar la ejecución y obtener estadísticas sobre el rendimiento.

4.2. LÓGICA DE PROCESAMIENTO

El motivo principal por el que se ha decidido usar scripts bash, para gestionar la lógica de procesamiento de la aplicación, es la idoneidad de esta técnica para invocar múltiples comandos independientes. Además, algunas de las utilidades proporcionadas por Sphinx-4, como es el caso del entrenamiento, están especialmente diseñadas para ser ejecutadas mediante scripts.

A continuación se expondrán, uno a uno, los scripts que integran el bloque de lógica de procesamiento de la aplicación:

- **`transcript.sh`**

Este es el script más importante de todos ya que es el que primero se ejecuta. Su importancia se debe, además, a que es el script que mayor número de funciones ejecuta, algunas de ellas críticas para la posterior ejecución de la aplicación. El proceso es el siguiente:

1. Una vez que Xplico haya creado los ficheros de audio a partir de los flujos RTP interceptados, el módulo de transcripción podrá acceder a ellos cuando sea necesario. Para ello, el script necesitará los siguientes parámetros:

- Como parámetros de entrada, el identificador SIP de cada interlocutor.
- Como parámetro de salida, el nombre que se quiera dar a la conversación.

2. El primer problema que se presenta en este punto es el hecho de que los ficheros de audio generados por Xplico tienen formato .mp3, mientras que Sphinx-4 sólo es capaz de procesar ficheros de audio en formato .wav, canal mono y PCM 16-bit. Por lo tanto, se hace necesario la utilización de un programa de conversión de formatos.

Inicialmente se pensó en ***Sox***, que es una aplicación muy versátil y potente para realizar transformaciones sobre ficheros de audio. Pero, precisamente, no ofrecía soporte en lo que se refiere a convertir ficheros de audio de formato .mp3 a .wav, debido a que mp3 no es un formato licenciado.

En consecuencia, hubo que buscar otra herramienta que permitiese llevar a cabo la transformación. Tras observar varias herramientas distintas, finalmente se decidió utilizar la aplicación ***ffmpeg***, la cual permite realizar dicha transformación de manera simple y rápida.

3. Una vez que contamos con los ficheros de audio en formato .wav, esta vez si se hace uso de la herramienta ***Sox*** para llevar a cabo la unión de ambos ficheros (conservando los originales) en uno sólo (estéreo), para ofrecer al usuario de la aplicación la posibilidad de escuchar el audio de cada interlocutor por separado, así como el de la conversación completa (en este caso, se escucha a un interlocutor por cada canal).

4. El siguiente paso es el más importante de todos y donde reside el grueso de la aplicación. Se trata de la ejecución del programa Java que se encarga de la transcripción de los ficheros de audio, mediante la librería de Sphinx-4.

Lo primero de todo será comprobar si se dispone de modelos acústicos entrenados para alguno de los interlocutores, ya que si fuera así, se usarían para aumentar la precisión en la transcripción.

Para ejecutar dicho código, es necesario pasar como parámetros:

- El *.jar* de Sphinx4 y el del modelo acústico (que puede ser el del interlocutor ya entrenado, si se dispone de él, o el del modelo acústico por defecto).

- También es necesario pasar el fichero `config.xml` que contiene la configuración de los componentes que utilizará Sphinx-4 para la transcripción (configuración del modelo acústico, del modelo de lenguaje, el diccionario, decodificador, etc).

- El identificador SIP del interlocutor cuyo fichero de audio se quiere transcribir (el fichero de audio del interlocutor tiene como nombre el identificador SIP de ese interlocutor).

- Como parámetro de salida habrá que indicarle la ruta y el nombre del fichero de texto donde se quiere que almacene la transcripción.

5. En este punto, ya se habría ejecutado la parte que más recursos y tiempo consumiría de la aplicación. El siguiente paso consiste en unir las dos transcripciones anteriores (una por cada interlocutor) en una sola, ordenando las frases a partir de los timestamps generados por Sphinx-4.

6. En este punto, ya se dispone de la transcripción completa de la conversación. Por lo tanto el proceso de transcripción ha terminado. Sin embargo, para que el usuario pueda visualizarla y manipularla, es necesario insertar un nuevo elemento en la base de datos de Xplico, que haga referencia a dicha conversación y a los ficheros asociados a ella.

- **`audioSplit.sh`**

Este script se encarga de fragmentar el audio de cada interlocutor en partes más pequeñas (frases). Esto es necesario por dos motivos:

- En Sphinx-4, a la hora de llevar a cabo el proceso de entrenamiento del sistema, es necesario descomponer los ficheros de audio, que se van a utilizar para entrenar, en fragmentos más pequeños que representen frases independientes. Es decir, cada vez que se detecte un silencio en el audio se considera una nueva frase. Este es el procedimiento estándar que utilizan la mayoría de sistemas de reconocimiento de voz open source que incorporan la funcionalidad de entrenamiento.

- Para permitir al usuario escuchar el audio asociado a cada frase y, de esa manera, facilitarle el trabajo de corrección de la transcripción.

A continuación se explica de manera más detallada:

1. Es importante tener claro que este script puede ser ejecutado desde distintos puntos de la aplicación. Su función es similar en ambos casos, pero presenta alguna pequeña variación. Por lo tanto, habrá que indicarle, mediante paso de parámetros, desde donde está siendo llamado.

2. La primera vez, será llamado al pulsarse el botón *EDIT* desde el interfaz gráfico. Este botón redirige hacia una nueva ventana que permitirá editar el texto de la conversación. Para dicho fin, habrá que presentarle al usuario el audio fragmentado de cada locutor acompañando a la frase que les corresponda.

3. Para poder llevar a cabo su labor necesitará recibir como parámetros:

- El identificador SIP de cada interlocutor, para poder acceder a los ficheros de audio de cada uno.
- El fichero de texto que contiene la transcripción de la conversación.
- El *pad*, que es el margen temporal que se añadirá a cada fragmento de audio a la hora de realizar la fragmentación. Sería una cantidad pequeña, en torno a 500 milisegundos. Con él se pretende solventar la limitación, en cuanto a precisión se refiere, que presenta Sphinx-4 a la hora de generar las marcas de tiempo o *timestamps*.

4. Lo primero de todo será crear las carpetas que almacenarán el audio fragmentado de cada interlocutor (y los ficheros de entrenamiento), si estos no existían con anterioridad. La estructura de directorios es la siguiente:

- Los ficheros de audio fragmentado se almacenarán en */users/id_sip_interlocutor/wav/*
- Los ficheros de entrenamiento se almacenarán en */users/id_sip_interlocutor/etc/*

5. Una vez hecho eso, se analizará el fichero de texto que contiene la transcripción de la conversación, cogiendo de cada línea los *timestamps* (de inicio y final de la frase) y el nombre del interlocutor. Mencionar en este punto que cada línea de la transcripción está compuesta por (y en este orden):

- Un *timestamp* de inicio y otro de final, que indican la duración de la frase.
- El nombre del interlocutor que interviene en esa frase.
- La transcripción de la frase.

Al *timestamp* de inicio se le restará el *pad* (si la resta no da negativa) para que empiece un poco antes. Al *timestamp* de final le sumará el *pad*. Al realizar estas operaciones, tanto el *timestamp* de inicio como el de final tienen valores distintos a los originales. Por lo tanto, habrá que sustituir los *timestamps* de la transcripción por estos nuevos.

A continuación se muestran unas líneas que representan un fragmento de una transcripción de ejemplo:

0.20 1.11 # Tom # hi my name is tom
1.53 3.02 # Mark # hi tom my name is mark how are you

Como se puede apreciar, en la transcripción generada por Sphinx-4 se omiten signos de puntuación y todo el texto se muestra en minúsculas.

6. Con la ayuda de la herramienta ***Sox*** se procede a fragmentar el audio, que tendrá la duración indicada por los nuevos *timestamps*, y que se almacenará en la carpeta del interlocutor que corresponde a la frase seleccionada con anterioridad.

Como nota final aclarar que, si no fuera llamado desde el botón *EDIT* sino desde el botón SAVE, las únicas diferencias consistirían en que ya no sería necesario crear las carpetas para almacenar los fragmentos, pues ya existirían, y que el *pad* sería nulo (se le pasaría como parámetro *pad* un 0, en vez de 0.5), debido a que cuando el script es llamado desde el botón *SAVE* lo que se pretende es guardar los cambios realizados por el usuario, que se supone que ha corregido la transcripción.

- **`prepare_training.sh`**

Este script se encarga de generar todos los ficheros de texto que Sphinx-4 requiere para ejecutar el entrenamiento de un interlocutor, a saber:

- `training.txt`: contiene la transcripción de la conversación en un formato especial, eliminando timestamps, signos de puntuación, mayúsculas y todo aquello que no sea texto puro.

- `training.fileids`: fichero de control en el que se almacenan el identificador de cada fragmento de audio.

- `training.transcription`: fichero que contiene el texto de la transcripción (a partir del texto proporcionado por `training.txt`) frase a frase, acotando cada una mediante el formato *<s></s>*.

- `training.dic`: fichero que contiene el diccionario que posee la pronunciación de cada palabra que aparece en el fichero `training.txt`.

Este script se ejecuta cuando el usuario pulse el botón *SAVE* para guardar los cambios que ha realizado sobre la transcripción.

Recibe como parámetros:

- Identificador SIP de cada interlocutor para poder acceder a sus ficheros de audio.
- Fichero de texto que contiene la transcripción de la conversación.

El proceso de ejecución del script es el siguiente:

1. Puede darse el caso de que el usuario quiera volver a modificar una transcripción que ya modificó con anterioridad. Esto dificulta las cosas, pues habrá que sobrescribir la parte correspondiente a dicha transcripción en los ficheros `training.transcription` y `training.fileids` (hay que tener en cuenta que dichos ficheros contienen datos de todas las transcripciones entrenadas con anterioridad, no solo de una). De modo que habrá que borrar todo lo relacionado con la transcripción previa, sin modificar lo de las demás transcripciones existentes. Esto puede hacerse con la ayuda del comando *"grep –v"*.

2. Independientemente de lo anterior, lo primero que hay que hacer es crear el fichero `training.txt`, a partir del fichero que contiene la transcripción de la conversación. Habrá que eliminar el nombre del interlocutor, los *timestamps*, signos de puntuación, etc, para quedarnos solamente con el texto de la misma.

3. A partir del fichero anterior, se crea t`raining.transcription`, con el siguiente formato de línea:

<s>texto aquí</s> (nombre_interlocutor/nombre_conversacion/id)

También se crea `training.fileids` con el siguiente formato de línea:

nombre_locutor/nombre_conversacion/id

4. Ya sólo queda crear el diccionario. Para ello, tendremos que ejecutar un script de Python proporcionado por el grupo CMU Sphinx, que se encarga de crearlo a partir del fichero `training.txt`.

Este script de Python presenta la peculiaridad de que necesita conectividad a Internet para su ejecución, ya que debe establecer conexión con los servidores de CMU Sphinx para realizar sus funciones.

Pueden crearse entradas repetidas en el diccionario, es decir, distintas pronunciaciones para una misma palabra, las cuales eliminaremos, de nuevo gracias al comando *"grep –v"*

Todo el proceso ha de realizarse dos veces, una por cada interlocutor.

- **`timestamps.sh`**

A modo de resumen, este script se encarga de modificar los *timestamps* en el fichero que contiene la transcripción, cuando el usuario, desde el interfaz gráfico, pulsa los botones (flechas) para disminuir o aumentar la duración de un fragmento de audio concreto.

Recibe como parámetros:

- El identificador SIP del interlocutor al que corresponde la frase cuya duración está siendo modificada.

- El nombre del fragmento de audio (que en el momento de ejecutarse este script, ya existiría, pues debería haber sido generado con anterioridad por el

script `audioSplit.sh`) que se corresponde con la frase que está siendo modificada.

- El fichero que contiene la transcripción de la conversación.

- El *timestamp* original y el *timestamp* que le sustituirá, que será el original más/menos (dependiendo de si se quiere aumentar o disminuir la duración) un margen concreto.

El proceso de ejecución del script es el que sigue:

1. Sustituir el *timestamp* antiguo por el nuevo en el fichero que contiene la transcripción (lo sobreescribe).

2. Filtrar la línea a la que pertenece el nuevo *timestamp*, en el fichero que contiene la transcripción.

3. Extraer de esa línea el *timestamp* de inicio y el de final.

4. Utilizar ***Sox*** para recortar un fragmento, con duración la de los timestamps, del fichero de audio del interlocutor. Ese fragmento sustituirá al que ya existía y que tiene su mismo nombre.

- **`training.sh`**

Este script es el encargado de llevar a cabo todo el proceso de entrenamiento de un interlocutor concreto. Para ser más exactos, este proceso se denomina adaptación del modelo acústico [19].

Adaptar un modelo acústico ya existente, permite lograr resultados más robustos que con el propio entrenamiento, ya que este último consiste en crear un modelo acústico adaptado desde cero.

Con este proceso se pretende adaptar el modelo acústico a las características de las conversaciones que se quieren transcribir, tales como la voz del interlocutor, el entorno en el que se realiza la comunicación, el canal de comunicación, etc.

Se ejecutará cuando se pulse el botón *TRAIN* del interfaz gráfico. Este botón estará inhabilitado hasta que se pulse el botón *SAVE*, que guarda los cambios realizados por el usuario. Esto es así porque la adaptación es un proceso delicado y no se debería ejecutar hasta que la transcripción haya sido corregida.

Con proceso delicado nos referimos al hecho de que si lo que se quiere es mejorar la precisión del sistema a la hora de transcribir los ficheros de audio de un determinado interlocutor, es necesario que, tanto la transcripción como la duración de los fragmentos de audio (los cuales se utilizan para este proceso de adaptación, y no el fichero de audio del interlocutor) sean lo más precisos posible. Ya que, si no lo son, lo que se estaría haciendo es empeorar el rendimiento del sistema, consiguiendo el efecto contrario al deseado.

También hay que señalar un aspecto importante, que es el sobreentrenamiento (o sobreadaptación). En todo sistema de reconocimiento de voz hay un eterno dilema que

consiste en lo siguiente: A medida que se aumenta la cantidad de datos a procesar por el sistema, se reduce el rendimiento de éste. Es algo que hay que asumir y cuya única solución reside en que los desarrolladores incorporen nuevas técnicas que potencien la capacidad de procesamiento y optimicen el rendimiento del sistema.

Por lo tanto, se recomienda entrenar sólo si se dispone del tiempo y los recursos necesarios como para generar transcripciones de calidad.

Antes de comenzar a explicar el proceso de adaptación del modelo acústico, es necesario copiar desde la carpeta de ***SphinxTrain*** a la carpeta ***bin*** (donde se encuentran los scripts) los siguientes ejecutables: *bw, map_adapt, gauden_counts, mixw_counts, mllr_matrix, mllr_solve,* y *tmat_counts.*

Indicado lo anterior, se procede a describir el proceso de ejecución del script:

1. Cada vez que se ejecute este script, con el fin de adaptar el modelo acústico existente a las características de un interlocutor concreto, se eliminará el modelo acústico adaptado perteneciente a dicho interlocutor, en caso de que existiese con anterioridad. Esto se debe a que el script utiliza todos los fragmentos de audio y ficheros de entrenamiento que un interlocutor tenga almacenados en su carpeta. Como éstos se conservan, cada vez que se ejecuta el script se crea desde cero el modelo acústico adaptado.

2. El primer paso a la hora de llevar a cabo la adaptación es generar los ficheros que contienen los vectores con características acústicas de cada fragmento de audio. Este proceso generará un fichero con extensión *.mfc* por cada fragmento de audio con extensión *.wav*.

3. A continuación hay que convertir el fichero *mdef* del modelo acústico a formato texto, para que pueda ser procesado por ***SphinxTrain***, usando para ello la herramienta ***PocketSphinx***.

4. El siguiente paso consiste en acumular estadísticas de los datos disponibles para la adaptación. Para ello hay que utilizar el ejecutable llamado "*bw*" de ***SphinxTrain***.

Los argumentos pasados al ejecutar *bw* deben coincidir con los parámetros definidos en el fichero *feat.params,* que se encuentra en la carpeta del modelo acústico.

También hay que copiar el fichero llamado *fillerdict* a la carpeta del modelo acústico y renombrarlo con el nombre *noisedict*.

Como nota final resaltar que el modelo acústico utilizado tiene que ser continuo, no semi-continuo.

5. Para realizar la adaptación propiamente dicha, se ejecutarán *mllr_solve* y *map_adapt* que son dos herramientas que permiten actualizar el modelo acústico original con los datos adaptados del interlocutor.

6. Para finalizar, el último paso consisten en crear un fichero *jar* del modelo acústico adaptado, dándole el nombre del interlocutor, y situarlo en la carpeta *lib* (que es donde Sphinx-4 almacena todos los *jars* de modelos acústicos) de Sphinx-4.

4.3. INTERFAZ WEB

Debido a que la aplicación debe integrarse con el software de Xplico, el lenguaje de programación utilizado para el diseño del interfaz web será PHP, y el framework CakePHP [20].

A continuación se describirán, uno a uno, los componentes que integran el Modelo-Vista-Controlador de la aplicación:

- ***Base de datos***

Se utilizará la base de datos proporcionada por Xplico, la cual utiliza ***SQLite3*** como programa gestor. Por lo tanto, primero se tendrá que diseñar las tablas de la aplicación y, posteriormente, integrarlas con las ya existentes de Xplico.

Estará compuesta por dos tablas: "*Conversations*" y "*Streams*"

- La tabla "*Conversations*" cuyos campos son:
  - Identificador de la conversación (clave primaria con auto-incremento).
  - Nombre de la conversación (índice)
  - Fecha de generación.
  - Interlocutores que intervienen en ella.

- La tabla "*Streams*" contendrá las referencias de todos los ficheros asociados a la conversación (audio de cada interlocutor, audio de la conversación, transcripción de la conversación). Sus campos son:
  - Identificador del fichero (clave primaria con auto-incremento)
  - Identificador de la conversación a la que pertenece (clave foránea)
  - Nombre del fichero (índice)
  - Tipo de fichero: audio o texto

- ***Vistas***

La capa vista de CakePHP representa el modo en que la aplicación se comunica con los usuarios.

Los ficheros de vista de CakePHP están escritos en PHP y tienen la extensión .ctp (CakePHP Template) por defecto. Estos ficheros contienen toda la lógica de representación necesaria para mostrar al usuario, de una manera inteligible y sencilla, los datos recibidos del controlador.

Los ficheros de vista se almacenan en el directorio */app/views/*. A modo de ejemplo, el fichero de vista para la acción *view()* del controlador *Conversations* se encontrará en */app/views/conversations/view.ctp*.

A continuación se describe el diseño de cada una de las vistas que integran la aplicación:

- **`index.ctp`**

  Es la primera vista de la aplicación. En ella se representan los datos proporcionados por la acción *index()* definida en el Controlador.

  Es bastante simple y muestra, básicamente, una tabla con todas las conversaciones (interceptadas por Xplico) que hay disponibles en ese momento.

  A continuación se muestra una imagen de ejemplo de la vista:

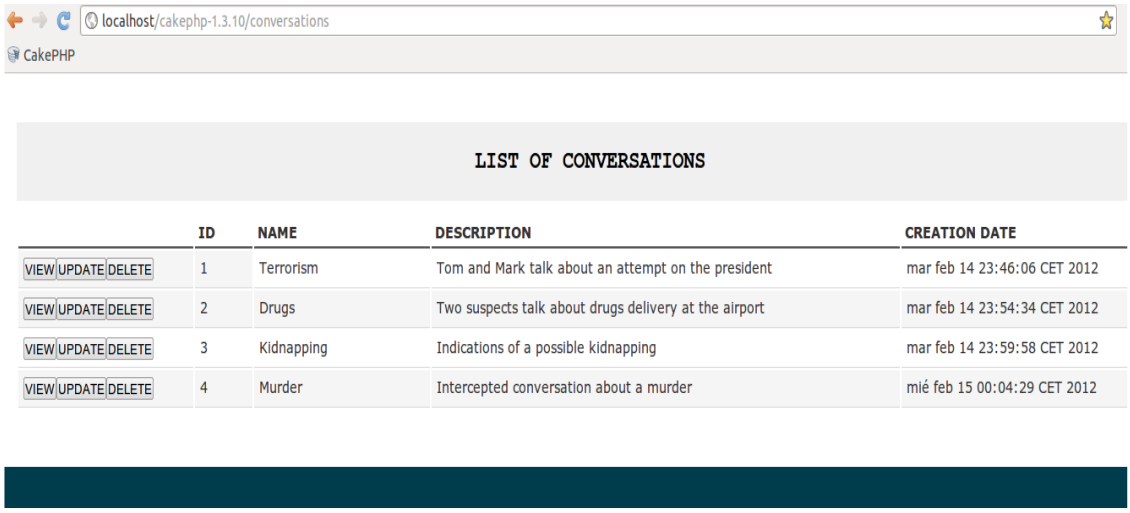

Figura 4.2: Ejemplo de ejecución de la vista index.ctp

Entre la información que cada conversación muestra al usuario se encuentran los siguientes campos: El nombre que el usuario ha asignado a la conversación (o el que ha asignado Xplico, si la conversación no ha sido aún manipulada), una breve descripción, y la fecha de creación.

También se muestran, al lado de cada conversación, tres botones:

- *UPDATE*: Al pulsarse nos redirige directamente hacia la vista `update.ctp` para permitir editar los campos de información asociada a la conversación.

- *VIEW*: Al pulsarse redirige hacia la vista `view.ctp`, para poder visualizar y analizar en profundidad la transcripción de la conversación.

- *DELETE*: Este botón permite eliminar la conversación y todos los elementos asociados a ella, tanto de la vista como de la base de datos.

- **`update.ctp`**

Esta es la vista más simple de todas. Muestra al usuario los datos transferidos por la acción *update()* definida en el Controlador.

A continuación se muestra una imagen de ejemplo de la vista:

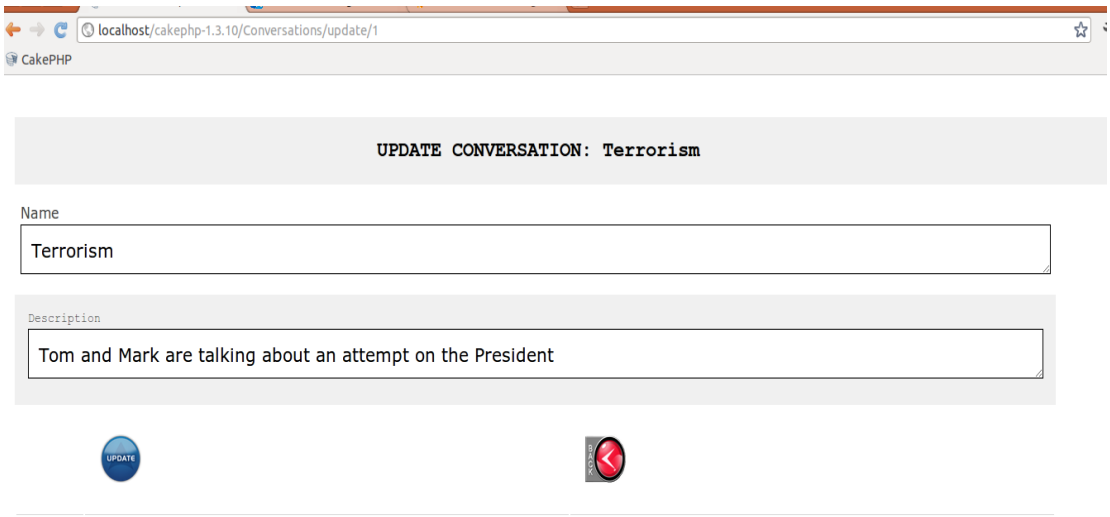

Figura 4.3: Ejemplo de ejecución de la vista update.ctp

Se compone de varios campos de texto que permiten al usuario modificar la información asociada a la conversación, tal como el nombre de la conversación, o la descripción.

Por último, dispone de un botón, denominado *UPDATE*, que permite actualizar los datos modificados por el usuario en la base de datos. Al pulsarlo, además de lo anterior, redirige a la vista `index.ctp`, la cual mostrará la conversación con los nuevos datos actualizados.

- **`view.ctp`**

A esta vista se llega desde la vista anterior, `index.ctp`, al pulsar el botón *VIEW*. En ella se muestran en detalle los distintos elementos asociados a la conversación que se ha seleccionado.

A continuación se muestra una imagen de ejemplo de la vista:

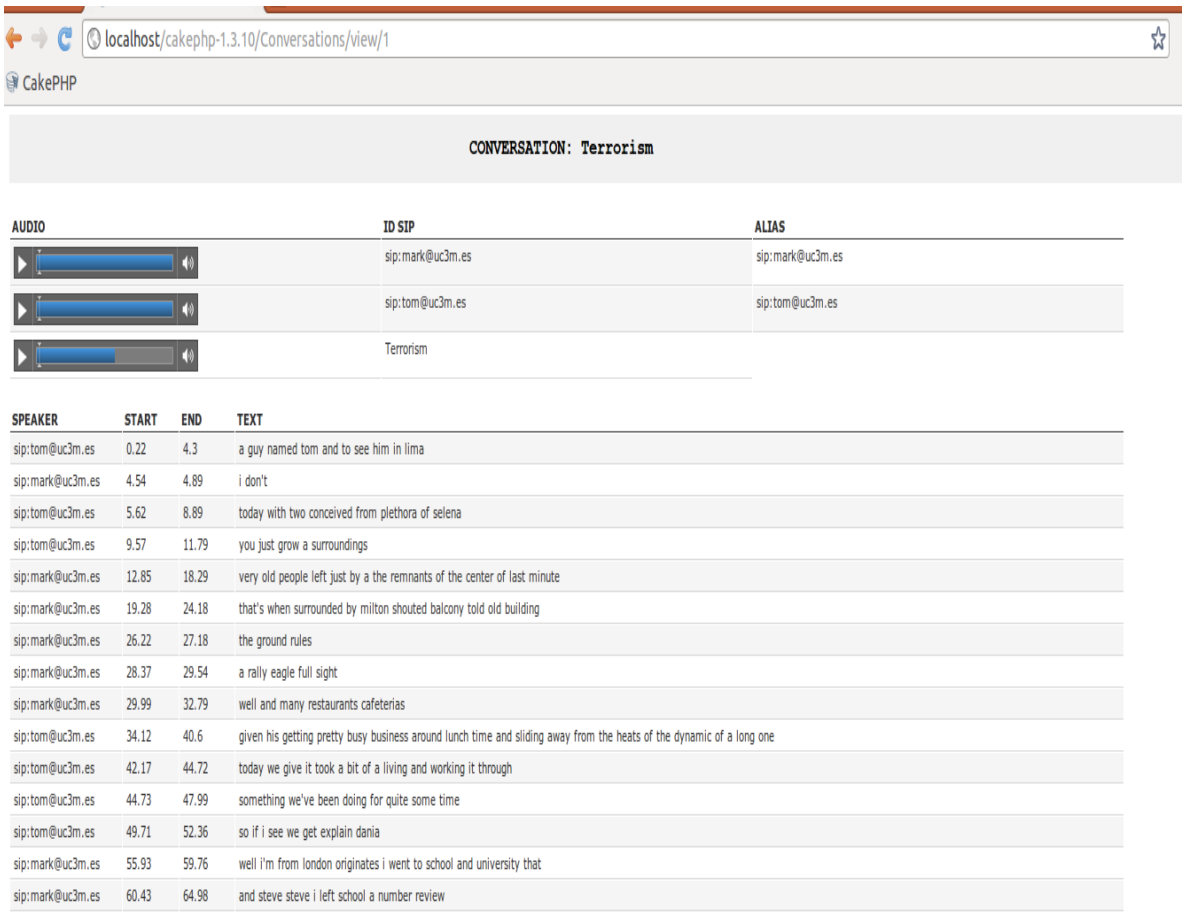

Figura 4.4: Ejemplo de ejecución de la vista view.ctp

Los primeros elementos que se muestran son los ficheros de audio de cada interlocutor y el de la conversación. Todos ellos permiten ser reproducidos gracias a la tecnología HTML5. El objetivo que se persigue permitiendo reproducir dichos ficheros de audio es el de facilitar el trabajo al usuario, a la hora de realizar la corrección de la transcripción. De este modo, puede leer la transcripción al mismo tiempo que escucha la conversación, o a cada interlocutor por separado, según lo prefiera.

Al lado del reproductor de audio de cada interlocutor se mostrarán dos cajas de texto. Una de ellas contendrá el nombre del interlocutor (que será un identificador SIP asignado por Xplico). La otra estará vacía y permitirá al usuario asignar un alias a cada interlocutor, para facilitar su reconocimiento (ya que los identificadores SIP pueden ser poco intuitivos). Al lado del reproductor de la conversación sólo figurará una caja de texto, que contendrá el nombre de la conversación (asignado por Xplico).

A continuación se muestra, frase a frase y a modo de tabla, la transcripción de la conversación. Cada frase contendrá, en este orden:

- *Timestamps* de inicio y de final de la frase.
- Nombre del interlocutor que corresponde a esa frase.
- Texto asociado a esa frase.

Finalmente, se llegó a la conclusión de que en esta vista no debería ofrecerse al usuario la posibilidad de editar la transcripción directamente. Esto es así, porque es probable que, en la mayoría de los casos, el usuario quiera leer la transcripción pero decida no modificar nada.

Por lo tanto, si realmente se desea modificar el contenido de la transcripción, el usuario deberá pulsar el botón denominado *EDIT*, situado en la parte inferior de la vista. Este botón le conducirá a la siguiente y última vista, llamada `edit.ctp`, la cual le permitirá, esta vez si, modificar el contenido de la transcripción. A su vez, se ejecutará el script `audioSplit.sh` que se encargará de fragmentar el audio de cada interlocutor para mostrar dichos fragmentos al lado de la frase que les corresponda, facilitando la labor de corrección de la transcripción al usuario.

- **`edit.ctp`**

A esta vista se llega cuando el usuario pulse el botón *EDIT*, situado en la vista anterior, `view.ctp`.

Esta vista es, sin duda, la más completa de todas. En ella se ofrecen funcionalidades como editar el texto de la conversación y entrenar el sistema con los datos (audio+transcripción) de los interlocutores.

A continuación se ofrece una imagen de ejemplo de la vista:

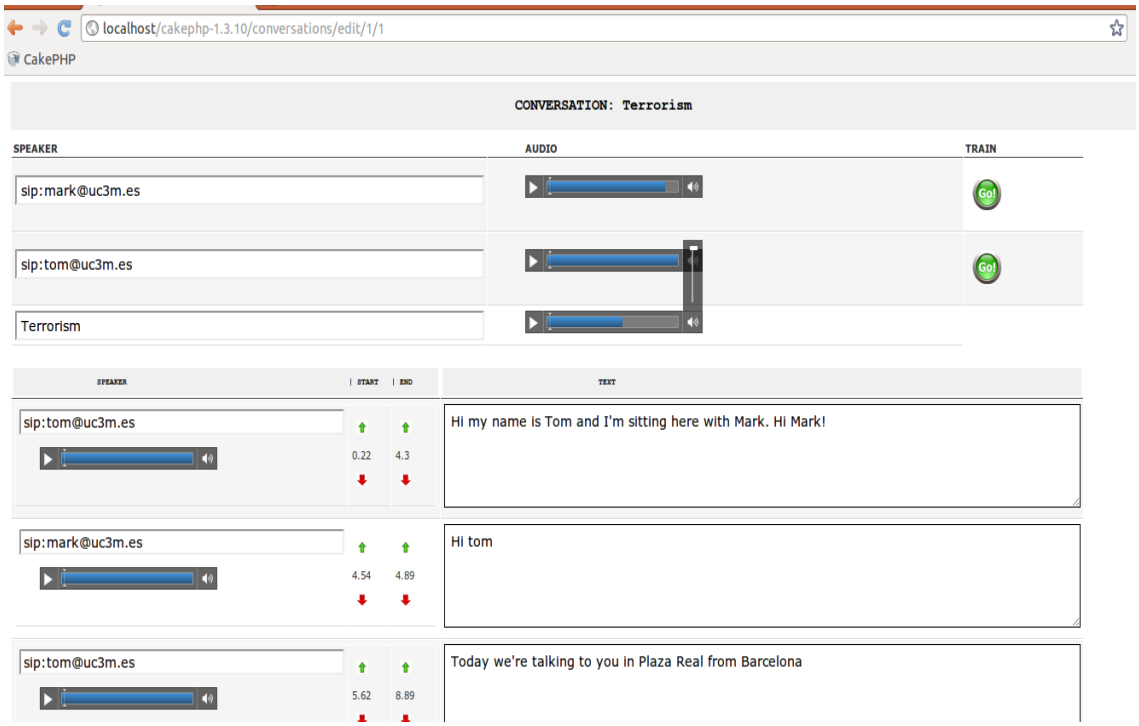

Figura 4.5: Ejemplo de ejecución de la vista edit.ctp

Lo primero que nos encontramos, al igual que en la vista anterior, son los reproductores de audio de cada interlocutor y el de la conversación. Al lado de cada uno de ellos se encuentra una caja de texto que contendrá el alias de cada locutor y, en el caso de la conversación, el nombre de ésta. Al lado del reproductor de audio de

cada interlocutor se encuentra, además de la caja de texto que contiene el alias, un botón de entrenamiento, denominado *TRAIN*, el cual ejecutará, al ser pulsado, el script `training.sh`.

Posteriormente se muestra, frase a frase y a modo de tabla, la transcripción de la conversación. Cada frase contendrá, en este orden:

- Reproductor de audio que reproducirá el fragmento de audio asociado a esa frase.
- *Timestamps* de inicio y de final de la frase.
- Nombre del locutor que corresponde a esa frase.
- Texto asociado a la frase.

Como se puede apreciar, la estructura de esta vista es prácticamente idéntica a la de la vista anterior. Como novedades destacan:

- Posibilidad de editar el texto.
- Posibilidad de entrenar el sistema.
- Posibilidad de escuchar el audio fragmentado asociado a cada frase, para facilitar al usuario el proceso de corrección de la transcripción.

Por último, situado en la parte inferior de la vista se encuentra un botón, denominado *SAVE*, que permite guardar el texto editado por el usuario. Además de eso, ejecuta los siguientes scripts:

- `audioSplit.sh`, que volverá a recortar los fragmentos de audio (a partir del fichero de audio de cada interlocutor) necesarios para el entrenamiento, pero esta vez, sin añadirles *pad*, ya que se supone que los *timestamps* estarán bien ajustados por el usuario.

- `prepare_training.sh`, que generará los ficheros de texto necesarios para el entrenamiento, a partir del fichero de texto que contiene la transcripción corregida.

Una vez pulsado el botón *SAVE*, y sólo entonces, se habilitarán los botones de entrenamiento de interlocutor. Esto es así por lo que ya se explicó con anterioridad, sobre que el proceso de entrenamiento es algo delicado y sólo se debería llevar a cabo utilizando datos (transcripción+audio) de calidad, es decir, datos fiables corregidos por el usuario.

- ***Controller***

Generalmente, en CakePHP, los controladores son usados para manejar la lógica de un sólo modelo. Además, se nombran según el modelo que manejan, y se ponen siempre en plural.

Los controladores de la aplicación son sub-clases de la clase AppController de CakePHP, que a su vez extiende la clase principal Controller. La clase AppController suele ser definida en /app/app_controller.php y debe contener métodos que son compartidos entre todos los controladores de la aplicación.

Los controladores pueden tener cualquier cantidad de métodos, a los que normalmente se les llama acciones. La función de una acción consiste en desarrollar un proceso concreto y enviar la información a su vista para que pueda ser mostrada al usuario.

A continuación se explica el diseño del controlador de la aplicación desarrollada:

- ***ConversationsController***

  El objetivo del controlador es manejar la lógica del modelo denominado *Conversations*.

  Posee la propiedad *hasMany*, que le permite relacionarse con múltiples objetos *Streams*, ya que cada objeto *Conversation* está relacionado con varios objetos *Streams*.

  Aparte de lo anterior, su función consiste en realizar de intermediador entre las vistas y la base de datos, permitiendo la comunicación entre ambas.

  Implementa una acción (método) por cada vista, a saber:

  - *index()*: Es la primera acción que se ejecuta. Accede a la base de datos y obtiene todos los objetos *Conversation* almacenados en ésta, para enviárselos, posteriormente, a la vista `index.ctp`.

  - *view():* Esta acción es llamada desde la vista `index.ctp`. Recibe como parámetro el identificador de un objeto *Conversation* concreto y, a partir de él, obtiene la conversación que corresponde. Después se lo envía a su vista correspondiente, `view.ctp`.

  - *update():* Esta acción puede ser llamada por dos vistas: `index.ctp` y `update.ctp`.

    Si es llamada por `index.ctp` simplemente tendrá que obtener la conversación cuyo identificador coincida con el proporcionado por la vista, y enviarlo a la vista `update.ctp`.

    Si es llamada por `update.ctp` (su vista asociada) será para que actualice en la base de datos la información asociada a la conversación, ya que el usuario podría haberla modificado.

  - *edit():* Esta acción puede ser llamada desde dos vistas: `view.ctp` y `edit.ctp`.

    Si es llamada por la primera, será cuando se haya pulsado el botón *EDIT*. Lo primero que hará será ejecutar el script `audioSplit.sh`, a partir de los datos enviados por la vista mediante petición *POST*. Posteriormente, actualizará el alias de cada interlocutor en la base de datos.

    Desde la vista `edit.ctp` puede ser llamada de tres formas distintas:

▪ Al pulsarse el botón *SAVE*: en este caso lo que se hará es, por este orden:

- Actualizar el fichero de texto que contiene la transcripción, con los cambios realizados por el usuario.
- Ejecutar `audioSplit.sh` con *pad* 0.
- Ejecutar `prepare_training.sh`.

▪ Al pulsarse el botón *TRAIN* de un interlocutor, lo único que hace es ejecutar el script `training.sh`, pasándole como parámetro el nombre de ese interlocutor.

▪ Al pulsarse cualquier flecha de los *timestamps*: en este caso lo que se hará es, por este orden:

- Primero, tiene que comprobar si la flecha pulsada es del *timestamp* de inicio o del *timestamp* de final de la frase. Además de eso, debe comprobar si la flecha pulsada es para aumentar o para disminuir el *timestamp*. Teniendo esto en cuenta, hay cuatro combinaciones posibles. Todo esto sirve, simplemente, para calcular el nuevo *timestamp* a partir del antiguo.
- Ejecutar el script `timestamps.sh`.

CAPÍTULO 5: EVALUACIÓN DEL SISTEMA DESARROLLADO

En este capítulo se describen en detalle los experimentos realizados y los resultados obtenidos, referidos al software de reconocimiento de voz Sphinx-4. Dichos experimentos se han llevado a cabo utilizando un modelo acústico independiente del locutor, con capacidad para reconocer amplios vocabularios en inglés (el más amplio que permite Sphinx-4), y adaptado a las características del canal (comunicación VoIP). Es decir, se ha elegido el modelo acústico que más se adapta a las necesidades del proyecto, de entre todos los disponibles.

Partiendo de esa base, se han realizados varios experimentos atendiendo a diferentes variables:

1. Para estimar de la manera más precisa posible la **precisión** del sistema de reconocimiento de voz, a la hora de realizar la transcripción de cada conversación, se realizaron las siguientes pruebas:

- Transcribir distintas conversaciones cuyos interlocutores sean desconocidos para el sistema, es decir, que no se haya entrenado con sus voces. En este caso, la tasa de acierto del sistema se ha situado en el rango del 30-50%, dependiendo de la voz que se intenta reconocer. Cuanto más se parezca la voz que se quiere reconocer a alguna de las usadas en el entrenamiento del modelo acústico, mayor será la tasa de acierto.

- Transcribir diferentes conversaciones cuyos interlocutores sean conocidos para el sistema, es decir, que haya sido entrenado con sus voces, pero no con ese diálogo concreto. En este caso, la tasa de acierto va incrementándose gradualmente, a medida que se aumentan los datos de entrenamiento de cada voz. Es decir, partiendo del rango del 30-50% anterior, cuantas más horas de grabación de una voz concreta se utilicen para entrenar el sistema, la tasa de acierto de éste a la hora de reconocer esa misma voz irá aumentando proporcionalmente. Lo normal es que se llegue a conseguir una tasa de acierto en torno al 50-70%, si se ha entrenado el sistema con una cantidad de datos significativa. No hay una fórmula matemática que diga cuántas horas de grabaciones pueden conseguir una tasa de acierto X. Depende de varios factores, de modo que la mejor manera de hacerlo es ir probando hasta que se encuentre un límite en el que la mejora se haga prácticamente inapreciable (generalmente, la tasa de acierto no superará el 70%).

- Transcribir varias conversaciones cuyos interlocutores sean conocidos para el sistema, es decir, que haya sido entrenado con sus voces, y además, con ese diálogo concreto. Esta situación es en la que se obtiene la mayor tasa de acierto del sistema, del rango de 80-90%, si bien es inútil para el caso concreto del proyecto actual, ya que difícilmente se encontrarán dos conversaciones exactamente iguales entre dos personas concretas. Simplemente, se menciona para señalar cual sería la situación de máximo potencial del sistema.

2. En lo referente al **consumo de recursos** del sistema, hay que destacar que la velocidad de Sphinx-4 durante proceso de reconocimiento de voz es bastante lenta, de modo que no está diseñado para ejecutarse en tiempo real. Más concretamente, se han realizado

diversas pruebas con ficheros de audio de distintas duraciones, hasta un máximo 5 minutos, obteniéndose las siguientes conclusiones:

- Como regla general, Sphinx-4 tarda aproximadamente el doble de tiempo que la duración del fichero de audio en realizar la transcripción. Es decir, si el fichero de audio tiene una duración de 2 minutos por ejemplo, Sphinx-4 tardaría en torno a 4 minutos en realizar la transcripción.

- Debido a que se analizan los ficheros de audio de cada interlocutor por separado, el tiempo total que tarda el sistema en generar la transcripción de la conversación completa, será la suma del tiempo de transcripción del audio de cada interlocutor. Siguiendo con el ejemplo anterior, si los ficheros de audio de cada interlocutor tienen una duración de 2 minutos cada uno, el sistema tardará aproximadamente 8 minutos en realizar la transcripción de la conversación completa.

3. El **canal de comunicación** es otro aspecto a tener muy en cuenta. Se han realizado pruebas para comparar la precisión del reconocedor de voz, utilizando dos modelos acústicos distintos. Uno de ellos entrenado con grabaciones de voz a frecuencias de muestreo de 16 kHz (comunicación normal), y el otro entrenado con grabaciones de voz a frecuencias de muestreo de 8 kHz (comunicación VoIP). Las conversaciones que se usaron como test eran las mismas que las utilizadas en los experimentos anteriores, solo que ahora se disponía de una versión muestreada a 16 kHz (original) y otra re-muestreada a 8 kHz, por cada conversación.

   Los resultados reafirmaron las bases teóricas. Estas bases eran las que aportaba inicialmente el grupo CMU Sphinx y consistían en que, a menor frecuencia de muestreo (menor calidad de la señal), más datos de entrenamiento se necesitaban para lograr resultados similares a los obtenidos cuando se utilizaban frecuencias de muestreo superiores. Es decir, la precisión del reconocedor de voz era superior cuando se usaba el modelo acústico adaptado a frecuencias de 16 kHz, que cuando se usó el modelo acústico adaptado a frecuencias de 8 kHz. Concretamente y de manera aproximada, la mejora de la precisión rondaba en torno al 10%, para cada caso enumerado en el punto 1 (de este capítulo).

   No obstante, estos resultados obtenidos son sólo una referencia, ya que los dos modelos acústicos utilizados para la comparación, poseen más diferencias aparte de la tasa de muestreo. También difieren en el tamaño del *corpus* (cantidad de grabaciones y duración de éstas), además de otras características que sólo el grupo CMU Sphinx – los creadores de estos modelos acústicos – conocen.

Por último, en lo que se refiere a la usabilidad del interfaz web, se ha intentado que sea lo más simple e intuitivo posible, para facilitar la labor de corrección de la transcripción por parte del analista de la Policía. Para ello, cada vista del interfaz web se relaciona con una funcionalidad concreta, mostrando al usuario únicamente lo que desea ver en ese momento, evitando así sobrecargarle de información. A modo de ejemplo, si el usuario sólo desea ver la transcripción generada por el reconocedor de voz, pero no tiene intención de modificarla, no se le muestran herramientas de edición en esa misma vista, sino únicamente el contenido de la conversación.

CAPÍTULO 6: CONCLUSIONES Y TRABAJOS FUTUROS

6.1. CONCLUSIONES

Después de haberse realizado una profunda investigación, con el fin de conseguir implementar un sistema de reconocimiento de voz lo más eficiente posible, atendiendo a los requisitos y limitaciones del proyecto, las conclusiones que se obtienen son las siguientes:

1. Actualmente los sistemas de reconocimiento de voz disponibles presentan arquitecturas muy similares, basadas en los Modelos Ocultos de Markov. Desde el sistema más sofisticado del momento, el Dragon Naturally Speaking de Nuance, empresa líder del mercado de tecnologías de voz, hasta los sistemas, tanto comerciales como open-source, más simples, pasando por el sistema implementado en este proyecto. Por lo tanto, lo que marca la diferencia entre unos sistemas y otros no es tanto su arquitectura, sino la calidad acústica (voces) que se han usado para entrenarlo. En cuanto a calidad acústica se refiere, ésta depende de tres variables:

- Cantidad y variedad léxica de los datos de entrenamiento: Si se quiere reconocer vocabularios amplios, deben recogerse datos suficientes que abarquen todo el léxico objetivo.
- Variedad fonética de las voces: Si se desea reconocer voces distintas, el sistema debe ser entrenado con datos que cubran la variedad fonética objetivo.
- Adaptación al canal de comunicación: El sistema se debe entrenar con datos en las mismas condiciones acústicas que en las que después se desea decodificar.

2. Teniendo en cuenta lo anterior, se ha implementado un sistema de reconocimiento de voz para aplicaciones de dictado (vocabularios extensos), independiente del locutor, y adaptado a comunicaciones VoIP.

3. Además, dicho sistema podrá ser mejorado continuamente mediante la incorporación de modelos acústicos de mayor calidad (publicados periódicamente por el grupo CMU Sphinx), y/o mediante entrenamiento.

4. Cuando se realice su integración con Xplico, permitirá llevar a cabo de manera eficaz la tarea de analizar, pre-clasificar e indexar flujos de datos VoIP, de manera automática.

5. A pesar de los beneficios aportados por el nuevo sistema, seguirá siendo necesaria la intervención humana para solventar las limitaciones que presenta esta tecnología. Por lo tanto, los analistas de la Policía deberán supervisar los resultados generados por la aplicación.

6.2. TRABAJOS FUTUROS

A continuación se destacan los trabajos futuros que deben desarrollarse para mejorar ciertos aspectos de la aplicación:

1. Integración con Xplico: Un aspecto fundamental que queda por completarse es la integración del sistema de reconocimiento de voz con la plataforma Xplico. Dicha integración no se ha llevado a cabo debido a la complejidad y el costo temporal que supone. Por este motivo se ha diseñado e implementado una aplicación independiente, pero adaptada a los requisitos de la plataforma de interceptación legal de comunicaciones, para facilitar su futura integración.

2. Otro aspecto pendiente es el referido a la dependencia del sistema de reconocimiento de voz, con respecto al grupo CMU Sphinx, en lo que se refiere al proceso de entrenamiento. Como se mencionó anteriormente en el capítulo de "Diseño del sistema", durante el proceso de entrenamiento hay un paso en el que es necesario conectarse a los servidores de CMU Sphinx para poder generar el diccionario adaptado al texto de entrenamiento. Cuando se lleve a cabo la integración con Xplico, habrá que solventar esta limitación para poder garantizar la confidencialidad de los datos manejados por la Policía.

3. Por último, el aspecto en el que más se debe insistir y volcar esfuerzos, si se quiere mejorar el rendimiento del sistema de reconocimiento de voz, es en la elaboración de modelos acústicos de calidad, que cubran amplios vocabularios, permitan reconocer voces distintas sin problemas, y se adapten con facilidad a distintos canales de comunicación.

REFERENCIAS

[1] Jesús Martín Porras. Normativa y política de telecomunicación. Interceptación legal de comunicaciones. PDF. Fecha de última visita: 02/2012

[2] Sitio web del Proyecto INDECT. Fecha de última visita: 02/2012. http://www.indect-project.eu/

[3] Roberto Gutiérrez Gil. “Seguridad en VoIP: Ataques, amenazas y riesgos”. Universitat de Valencia. Fecha de última visita: 02/2012.

[4] IETF. Protocolo SIP. RFC 3261. Fecha de última visita: 02/2012.

[5] IETF. Protocolo RTP. RFC 3550. Fecha de última visita: 02/2012.

[6] IETF. Protocolo RTCP. RFC 3605. Fecha de última visita: 02/2012.

[7] IETF. Protocolo SDP. RFC 4566. Fecha de última visita: 02/2012.

[8] ITU-T Recommendation. Códec G.711. Fecha de última visita: 02/2012

[9] Software de análisis forense Xplico. Fecha de última visita: 02/2012. http://www.xplico.org/

[10] Sitio web de CMU Sphinx. Fecha de última visita: 02/2012. http://cmusphinx.sourceforge.net/sphinx4/

[11] Sitio web de Julius. Fecha de última visita: 02/2012. http://julius.sourceforge.jp/en_index.php

[12] Sitio web de HTK. Fecha de última visita: 02/2012. http://htk.eng.cam.ac.uk/

[13] Sitio web de ISIP. Fecha de última visita: 02/2012. http://www.isip.piconepress.com/projects/speech/index.html

[14] Sitio web de SIMON. Fecha de última visita: 02/2012. http://simon-listens.org/index.php?id=122&L=1

[15] Sitio web de MARF. Fecha de última visita: 02/2012. http://marf.sourceforge.net/

[16] Sitio web de openSMILE. Fecha de última visita: 02/2012. http://opensmile.sourceforge.net/

[17] Sitio web de SPRAAK. Fecha de última visita: 02/2012. http://www.spraak.org/

[18] Sitio web de iATROS. Fecha de última visita: 02/2012.
http://prhlt.iti.upv.es/page/projects/multimodal/idoc/iatros

[19] Tutorial sobre el proceso de adaptación acústica en Sphinx-4. Fecha de última visita: 02/2012.
http://cmusphinx.sourceforge.net/wiki/tutorialadapt

[20] Sitio web de CakePHP. Fecha de última visita: 02/2012.
http://book.cakephp.org/

ANEXOS

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor: Jesús Vallinot Sánchez

2.- Departamento: Ingeniería Telemática

3.- Descripción del Proyecto:

- Titulo: **Módulo de transcripción VoIP-texto para una plataforma de interceptación legal de comunicaciones**
- Duración (meses): **18**
- Tasa de costes Indirectos: 20%

4.- Presupuesto total del Proyecto (valores en Euros): 33.000
Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
Manuel Urueña Pascual		Ingeniero Senior	1	4.289,54	4.289,54	
Jesús Vallinot Sánchez		Ingeniero	6	2.694,39	16.166,34	
					0,00	
					0,00	
		Hombres mes	**7**	**Total**	20.455,88	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador portátil	670,00	100	18	60	201,00
					0,00
					0,00
					0,00
					0,00
					0,00
				Total	201,00

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
Paquete Microsoft Office		100,00
Abono transporte		480,00
Software open source Sphinx-		0,00
Framework CakePHP		0,00
	Total	580,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible,

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes**
Personal	20.456
Amortización	201
Subcontratación de tareas	0
Costes de funcionamiento	580
Costes Indirectos	4.247
Total	25.484